

Matematica delle Reti Neurali

Guida completa ed esplicita

Fondamenti · MLP · CNN · LSTM · Transformer · Ottimizzazione · Statistica Bayesiana

Ogni derivazione passo per passo

Ogni simbolo definito

Esempi numerici espliciti

Intuizioni geometriche

PARTE 0

Notazioni, Simboli e Prerequisiti

Ogni simbolo matematico usato nel documento viene definito qui, compresi quelli introdotti nelle parti avanzate. Non viene assunto nessun prerequisito oltre all'algebra di base delle scuole superiori.

0.1 Insiemi Numerici

Simbolo	Significato	Esempi
$\mathbb{N}$	Numeri naturali (interi non negativi)	$0, 1, 2, 3, \dots$
$\mathbb{Z}$	Numeri interi (positivi, negativi, zero)	$\dots, -2, -1, 0, 1, 2, \dots$
$\mathbb{R}$	Numeri reali (tutti i numeri sulla retta)	$-3.14, 0, \sqrt{2}, e, \pi$

$\mathbb{R}^n$	Spazio dei vettori colonna con n componenti reali	$(1.2, -0.5, 3.0)^T \in \mathbb{R}^3$
$\mathbb{R}^{m \times n}$	Spazio delle matrici con m righe e n colonne reali	Matrice 3×2 ha 6 elementi
$x \in S$	x appartiene all'insieme S	$3 \in \mathbb{N}$
$\forall$	"per ogni"	$\forall x \in \mathbb{R} : x^2 \geq 0$
$\exists$	"esiste almeno un"	$\exists x \in \mathbb{R} : x^2 = 4$
$\triangleq$	Definizione (il simbolo a sinistra viene definito come il simbolo a destra)	$\sigma(z) \triangleq \frac{1}{1+e^{-z}}$

0.2 Vettori e Operazioni

Cos'è un vettore

Un **vettore colonna** $\mathbf{x} \in \mathbb{R}^n$ è una lista ordinata di n numeri reali, scritti in colonna. Il pedice indica la componente:

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}, \quad x_i \in \mathbb{R} \text{ per ogni } i = 1, \dots, n$$

Un **vettore riga** si ottiene con la **trasposizione** $\mathbf{x}^T = (x_1, x_2, \dots, x_n)$. Il simbolo T sopra ribalta righe e colonne.

Norma euclidea (lunghezza del vettore)

La norma di $\mathbf{x}$, scritta $\|\mathbf{x}\|$ o $\|\mathbf{x}\|_2$, è la sua lunghezza geometrica:

$$\|\mathbf{x}\| \triangleq \sqrt{x_1^2 + x_2^2 + \dots + x_n^2} = \sqrt{\sum_{i=1}^n x_i^2}$$

Il simbolo $\sum_{i=1}^n$ significa "somma al variare di i da 1 a n ". È una scorciatoia:
 $\sum_{i=1}^3 x_i^2 = x_1^2 + x_2^2 + x_3^2$.

Prodotto scalare (dot product)

Il **prodotto scalare** tra $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ produce un numero (uno scalare), moltiplicando le componenti corrispondenti e sommando:

$$\mathbf{x} \cdot \mathbf{y} \triangleq \mathbf{x}^T \mathbf{y} = x_1 y_1 + x_2 y_2 + \dots + x_n y_n = \sum_{i=1}^n x_i y_i$$

INTERPRETAZIONE GEOMETRICA FONDAMENTALE

Vale sempre: $\mathbf{x}^T \mathbf{y} = \|\mathbf{x}\| \cdot \|\mathbf{y}\| \cdot \cos \theta$, dove θ è l'angolo tra i due vettori.

Quindi:

- Se $\mathbf{x}$ e $\mathbf{y}$ puntano nella stessa direzione ($\theta = 0^\circ$): $\cos \theta = 1$, prodotto scalare massimo
- Se sono perpendicolari ($\theta = 90^\circ$): $\cos \theta = 0$, prodotto scalare = 0
- Se puntano in direzioni opposte ($\theta = 180^\circ$): $\cos \theta = -1$, prodotto scalare negativo

Questa proprietà è la base del meccanismo di attenzione nei Transformer (Sezione 9).

ESEMPIO NUMERICO — PRODOTTO SCALARE

Siano $\mathbf{x} = (2, -1, 3)^T$ e $\mathbf{y} = (1, 4, -2)^T$:

$$\mathbf{x}^T \mathbf{y} = (2)(1) + (-1)(4) + (3)(-2) = 2 - 4 - 6 = -8$$

$$\|\mathbf{x}\| = \sqrt{4 + 1 + 9} = \sqrt{14} \approx 3.742, \quad \|\mathbf{y}\| = \sqrt{1 + 16 + 4} = \sqrt{21} \approx 4.583$$

$$\cos \theta = \frac{-8}{3.742 \times 4.583} \approx -0.466 \implies \theta \approx 117.7^\circ$$

I due vettori formano un angolo ottuso — puntano in direzioni "opposte".

Prodotto di Hadamard (elemento per elemento)

Dati $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, il simbolo $\odot$ indica la moltiplicazione *componente per componente*:

$$\mathbf{x} \odot \mathbf{y} = \begin{pmatrix} x_1 y_1 \\ x_2 y_2 \\ \vdots \\ x_n y_n \end{pmatrix}$$

Il risultato è ancora un vettore in $\mathbb{R}^n$. Non va confuso con il prodotto scalare, che produce invece un numero.

0.3 Matrici e Moltiplicazione

Definizione

Una matrice $\mathbf{W} \in \mathbb{R}^{m \times n}$ è una griglia rettangolare di numeri reali con m righe e n colonne. L'elemento alla riga i e colonna j si scrive W_{ij} o w_{ij} :

$$\mathbf{W} = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & & \ddots & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mn} \end{pmatrix}$$

Moltiplicazione matrice–vettore

Dati $\mathbf{W} \in \mathbb{R}^{m \times n}$ e $\mathbf{x} \in \mathbb{R}^n$, il prodotto $\mathbf{y} = \mathbf{W}\mathbf{x} \in \mathbb{R}^m$ si calcola così: la componente i -esima di $\mathbf{y}$ è il prodotto scalare tra la i -esima *riga* di $\mathbf{W}$ e il vettore $\mathbf{x}$:

$$y_i = \sum_{j=1}^n w_{ij} x_j = w_{i1}x_1 + w_{i2}x_2 + \cdots + w_{in}x_n, \quad i = 1, \dots, m$$

Nota bene: le dimensioni devono essere compatibili. Se $\mathbf{W}$ è $m \times n$, allora $\mathbf{x}$ deve essere $n \times 1$ (cioè avere n componenti) e il risultato ha m componenti.

ESEMPIO — MATRICE 2×3 PER VETTORE IN $\mathbb{R}^3$

$$\mathbf{W} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \in \mathbb{R}^{2 \times 3}, \quad \mathbf{x} = \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix} \in \mathbb{R}^3$$

$$\mathbf{y} = \mathbf{W}\mathbf{x} = \begin{pmatrix} 1 \cdot 1 + 2 \cdot 0 + 3 \cdot (-1) \\ 4 \cdot 1 + 5 \cdot 0 + 6 \cdot (-1) \end{pmatrix} = \begin{pmatrix} 1 + 0 - 3 \\ 4 + 0 - 6 \end{pmatrix} = \begin{pmatrix} -2 \\ -2 \end{pmatrix} \in \mathbb{R}^2$$

Trasposizione di una matrice

La trasposta $\mathbf{W}^T \in \mathbb{R}^{n \times m}$ si ottiene scambiando righe e colonne: $(W^T)_{ij} = W_{ji}$. In pratica, la prima riga diventa la prima colonna, la seconda riga diventa la

seconda colonna, ecc.

Derivate di operazioni matriciali — risultati usati nel backprop

Se $\mathbf{y} = \mathbf{W}\mathbf{x}$ e $\mathcal{L}$ è un numero scalare (la loss), allora:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \frac{\partial \mathcal{L}}{\partial \mathbf{y}} \cdot \mathbf{x}^T \in \mathbb{R}^{m \times n} \quad (\text{prodotto esterno})$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}} = \mathbf{W}^T \cdot \frac{\partial \mathcal{L}}{\partial \mathbf{y}} \in \mathbb{R}^n \quad (\text{trasposta di } \mathbf{W})$$

Queste formule si derivano componente per componente (vedi Sezione 4).

Vengono usate intensivamente nel backpropagation.

0.4 Calcolo Differenziale

La derivata come tasso di variazione

La derivata $f'(x)$ di una funzione $f: \mathbb{R} \rightarrow \mathbb{R}$ misura quanto velocemente cambia f al variare di x . Formalmente è il limite del rapporto incrementale:

$$f'(x) = \frac{df}{dx} \triangleq \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$

Geometricamente, è la pendenza della retta tangente al grafico di f nel punto x .

Derivate fondamentali usate nel documento

Funzione	Derivata	Come si ottiene
----------	----------	-----------------

c (costante)	0	Una costante non cambia
x^n	$n x^{n-1}$	Regola della potenza
e^x	e^x	Proprietà speciale di e
e^{ax}	$a e^{ax}$	Regola della catena: $(e^u)' = e^u \cdot u'$
$\ln x$	$1/x$ (per $x > 0$)	Inversa di e^x
$\ln(f(x))$	$f'(x)/f(x)$	Catena: $(\ln u)' = u'/u$

Regola del prodotto

Per $h(x) = f(x) \cdot g(x)$:

$$h'(x) = f'(x) g(x) + f(x) g'(x)$$

Regola del quoziente

Per $h(x) = f(x)/g(x)$ con $g(x) \neq 0$:

$$h'(x) = \frac{f'(x) g(x) - f(x) g'(x)}{[g(x)]^2}$$

Regola della catena — la più importante per le reti neurali

Se $h(x) = f(g(x))$ (funzione composta), allora:

$$h'(x) = f'(g(x)) \cdot g'(x) \quad \text{cioè: "derivata dell'esterna valutata nell'interiore"}$$

ESEMPIO — REGOLA DELLA CATENA

Sia $h(x) = e^{-x^2}$. Poniamo $g(x) = -x^2$ (funzione interna) e $f(u) = e^u$ (funzione esterna):

$$h'(x) = f'(g(x)) \cdot g'(x) = e^{-x^2} \cdot (-2x)$$

Gradiente — derivata per funzioni a più variabili

Per $f : \mathbb{R}^n \rightarrow \mathbb{R}$ (che prende un vettore e restituisce un numero), il **gradiente** è il vettore di tutte le derivate parziali. La derivata parziale $\partial f / \partial x_i$ tratta tutte le variabili tranne x_i come costanti:

$$\nabla_{\mathbf{x}} f = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix} \in \mathbb{R}^n$$

Il gradiente punta nella direzione di **massima crescita** di f . Muoversi nella direzione $-\nabla f$ riduce f — questa è l'essenza del gradient descent.

ESEMPIO — GRADIENTE DI UNA FUNZIONE SCALARE

Sia $f(x_1, x_2) = 3x_1^2 + 2x_1x_2 - x_2^2$. Le derivate parziali:

$$\frac{\partial f}{\partial x_1} = 6x_1 + 2x_2 \quad (\text{tratto } x_2 \text{ come costante})$$

$$\frac{\partial f}{\partial x_2} = 2x_1 - 2x_2 \quad (\text{tratto } x_1 \text{ come costante})$$

$$\nabla f = \begin{pmatrix} 6x_1 + 2x_2 \\ 2x_1 - 2x_2 \end{pmatrix}$$

Nel punto $(1, 2)$: $\nabla f = (10, -2)^T$. La funzione cresce più rapidamente nella direzione $(10, -2)$.

0.5 Probabilità Essenziale

Variabile aleatoria e distribuzione

Una **variabile aleatoria** X è una quantità il cui valore è incerto. La sua **distribuzione di probabilità** descrive quanto è probabile ogni valore. Per una variabile discreta: $P(X = k) = p_k \geq 0$ e $\sum_k p_k = 1$.

Valore atteso (media)

$$\mathbb{E}[X] = \sum_k k \cdot P(X = k)$$

Per funzioni di variabili aleatorie: $\mathbb{E}[f(X)] = \sum_k f(k) \cdot P(X = k)$.

Varianza

$$\text{Var}(X) = \mathbb{E}[(X - \mathbb{E}[X])^2] = \mathbb{E}[X^2] - (\mathbb{E}[X])^2$$

Distribuzione gaussiana (normale)

$X \sim \mathcal{N}(\mu, \sigma^2)$ significa che X segue una distribuzione gaussiana con media μ e varianza σ^2 . La densità di probabilità è:

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x - \mu)^2}{2\sigma^2}\right)$$

Logaritmo naturale — proprietà usate

Il logaritmo naturale $\ln$ (base $e \approx 2.718$) soddisfa:

- $\ln(e^x) = x$ e $e^{\ln x} = x$
- $\ln(ab) = \ln a + \ln b$
- $\ln(a/b) = \ln a - \ln b$
- $\ln(a^n) = n \ln a$
- $\ln(1) = 0, \ln x \rightarrow -\infty$ per $x \rightarrow 0^+, \ln x \rightarrow +\infty$ per $x \rightarrow +\infty$

0.6 Simboli Avanzati Usati nelle Parti Successive

Matrici speciali

Simbolo	Significato	Dove usato
$\mathbf{J}_f$	Jacobiana di $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$: matrice $m \times n$ con $(J_f)_{ij} = \partial f_i / \partial x_j$	Backprop, Normalizing Flows
$\mathbf{H}_f$ o $\nabla^2 f$	Hessiana di $f : \mathbb{R}^n \rightarrow \mathbb{R}$: matrice $n \times n$ con $H_{ij} = \partial^2 f / \partial x_i \partial x_j$	Ottimizzazione, EWC
$\lambda_i(\mathbf{A})$	i -esimo autovalore di $\mathbf{A}$: scalare tale che $\mathbf{A}\mathbf{v}_i = \lambda_i\mathbf{v}_i$ per qualche $\mathbf{v}_i \neq \mathbf{0}$	Analisi convergenza
$\mathbf{A} \otimes \mathbf{B}$	Prodotto di Kronecker: se $\mathbf{A} \in \mathbb{R}^{m \times n}$, risultato è $\mathbb{R}^{mp \times nq}$ con blocchi $a_{ij}\mathbf{B}$	K-FAC

$\text{tr}(\mathbf{A})$	Traccia: somma degli elementi diagonali $\sum_i A_{ii}$	Score matching
$\mathbf{A} \succeq \mathbf{0}$	$\mathbf{A}$ è semidefinita positiva: $\mathbf{v}^T \mathbf{A} \mathbf{v} \geq 0$ per ogni $\mathbf{v}$	Fisher Information
$[\mathbf{u}; \mathbf{v}]$	Concatenazione verticale di vettori: $\begin{pmatrix} \mathbf{u} \\ \mathbf{v} \end{pmatrix}$	LSTM, GRU
$\ \mathbf{A}\ _F$	Norma di Frobenius: $\sqrt{\sum_{ij} A_{ij}^2}$	Regolarizzazione

Notazione asintotica (complessità computazionale)

Simbolo	Significato	Esempio
$O(f(n))$	"Ordine di": cresce al massimo come $cf(n)$ per n grande	$O(n^2)$: quadratico
$\Omega(f(n))$	Cresce almeno come $f(n)$	$\Omega(n \log n)$
$\Theta(f(n))$	Cresce esattamente come $f(n)$ (sia upper che lower bound)	$\Theta(nd)$

Nelle sezioni di complessità, n indica tipicamente la lunghezza della sequenza, d la dimensione degli embedding, B il batch size.

Funzioni speciali

Simbolo	Definizione	Dove usato
$\mathbf{1}[P]$	Funzione indicatrice: 1 se la proposizione P è vera, 0 altrimenti	ReLU, masking
$\text{sign}(x)$	Segno: $+1$ se $x > 0$, -1 se $x < 0$, 0 se $x = 0$	L1, pruning
$\lfloor x \rfloor$	Parte intera inferiore (floor): massimo intero $\leq x$	Stride convoluzioni

$\lceil x \rceil$	Parte intera superiore (ceiling): minimo intero $\geq x$	Padding
$\text{softplus}(x)$	$\ln(1 + e^x)$: approssimazione morbida di $\max(0, x)$	Mamba, attivazioni
$\Phi(x)$	CDF gaussiana standard: $P(Z \leq x)$ con $Z \sim \mathcal{N}(0, 1)$	GELU
$\text{erf}(x)$	Funzione errore: $\frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$	GELU
δ_{ij}	Delta di Kronecker: 1 se $i = j$, 0 se $i \neq j$	Softmax, attention

Notazione probabilistica avanzata

Simbolo	Significato
$p(\mathbf{x} \theta)$ o $p_\theta(\mathbf{x})$	Distribuzione parametrizzata da θ
$\mathbf{x} \sim p$	" $\mathbf{x}$ è campionato dalla distribuzione p "
$\mathbb{E}_{\mathbf{x} \sim p}[f(\mathbf{x})]$	Valore atteso di $f(\mathbf{x})$ quando $\mathbf{x}$ è campionato da p
$D_{\text{KL}}(p q)$	KL divergence: $\mathbb{E}_p[\log(p/q)] \geq 0$, zero sse $p = q$
$H(p)$	Entropia di Shannon: $-\mathbb{E}_p[\log p] \geq 0$
$H(p, q)$	Cross-entropy: $-\mathbb{E}_p[\log q] = H(p) + D_{\text{KL}}(p q)$
$X \perp Y$	X e Y indipendenti: $p(X, Y) = p(X)p(Y)$
i.i.d.	Identicamente distribuiti e indipendenti

Numeri complessi (usati in RoPE e analisi spettrale)

$$i = \sqrt{-1}, \quad z = a + bi \in \mathbb{C}, \quad |z| = \sqrt{a^2 + b^2}$$

$$e^{i\theta} = \cos \theta + i \sin \theta \quad (\text{formula di Eulero})$$

$$\begin{aligned} \operatorname{Re}[z] &= a \text{ (parte reale),} & \operatorname{Im}[z] &= b \text{ (parte immaginaria)} \\ \bar{z} &= a - bi \text{ (coniugato),} & z\bar{z} &= |z|^2 \end{aligned}$$

Il prodotto di due numeri complessi in forma polare corrisponde a sommare gli angoli: $r_1e^{i\theta_1} \cdot r_2e^{i\theta_2} = r_1r_2e^{i(\theta_1+\theta_2)}$ — questa è la proprietà usata in RoPE per codificare le posizioni come rotazioni.

Processi stocastici (usati nei diffusion model e SGD)

Simbolo	Significato
$d\mathbf{W}_t$	Incremento di moto Browniano: $d\mathbf{W}_t \sim \mathcal{N}(\mathbf{0}, dt \cdot \mathbf{I})$
$d\theta = f\,dt + g\,d\mathbf{W}_t$	Equazione differenziale stocastica (SDE)
$\bar{\alpha}_t = \prod_{s=1}^t (1 - \beta_s)$	Prodotto cumulativo dei fattori di ruido (diffusion)

Notazione di sommatoria doppia e prodotto

$$\begin{aligned} \sum_{i=1}^m \sum_{j=1}^n a_{ij} &= \text{somma di tutti gli } mn \text{ elementi della matrice } \mathbf{A} \\ \prod_{t=1}^T x_t &= x_1 \cdot x_2 \cdots x_T \quad (\text{produttoria}) \\ \bigoplus_{i=1}^n \mathbf{v}_i &= [\mathbf{v}_1; \mathbf{v}_2; \dots; \mathbf{v}_n] \quad (\text{concatenazione}) \end{aligned}$$

Il Neurone Artificiale

1.1 Il Modello Matematico

Un neurone artificiale prende in ingresso un vettore $\mathbf{x} \in \mathbb{R}^n$ (gli n segnali di input) e produce un singolo numero (l'attivazione). Il calcolo avviene in due fasi:

DEFINIZIONE — NEURONE ARTIFICIALE

Fase 1 — combinazione lineare pesata (pre-attivazione):

$$z = w_1x_1 + w_2x_2 + \dots + w_nx_n + b = \mathbf{w}^T \mathbf{x} + b$$

dove $\mathbf{w} = (w_1, \dots, w_n)^T \in \mathbb{R}^n$ è il vettore dei **pesi** e $b \in \mathbb{R}$ è il **bias**.

Fase 2 — applicazione della funzione non-lineare:

$$a = f(z)$$

dove $f : \mathbb{R} \rightarrow \mathbb{R}$ è la **funzione di attivazione** e a è l'**attivazione**.

Il ruolo del bias

Il bias b è un termine costante aggiunto alla somma pesata. Senza di esso, quando $\mathbf{x} = \mathbf{0}$ (tutti gli input sono zero), si avrebbe sempre $z = 0$. Il bias permette di spostare il punto di "accensione" del neurone: cambia dove la funzione di attivazione viene valutata, aggiungendo flessibilità.

Geometricamente, $\mathbf{w}^T \mathbf{x} + b = 0$ definisce un **iperpiano** nello spazio $\mathbb{R}^n$. Per $n = 2$: è una retta del piano. Il neurone separa lo spazio in due regioni: quella dove $z > 0$ e quella dove $z < 0$.

Perché la non-linearità è indispensabile

Supponiamo di impilare due neuroni senza funzione di attivazione (o con attivazione identità $f(z) = z$). Il secondo neurone calcola:

$$a^{[2]} = \mathbf{w}^{[2]T} (\underbrace{\mathbf{W}^{[1]} \mathbf{x} + \mathbf{b}^{[1]}}_{\mathbf{a}^{[1]}}) + b^{[2]} = \underbrace{\mathbf{w}^{[2]T} \mathbf{W}^{[1]}}_{\text{nuova matrice}} \mathbf{x} + \underbrace{\mathbf{w}^{[2]T} \mathbf{b}^{[1]} + b^{[2]}}_{\text{nuovo bias}}$$

Il risultato è ancora una trasformazione lineare di $\mathbf{x}$. Non importa quanti layer lineari si impilano: il comportamento complessivo è sempre equivalente a un singolo neurone lineare. La funzione di attivazione non-lineare rompe questa degenerazione.

1.2 Funzioni di Attivazione

Sigmoid $\sigma(z)$

$$\sigma(z) \triangleq \frac{1}{1 + e^{-z}}$$

Comprime qualsiasi numero reale nell'intervallo aperto $(0, 1)$. Per $z \rightarrow +\infty$: $\sigma(z) \rightarrow 1$. Per $z \rightarrow -\infty$: $\sigma(z) \rightarrow 0$. Il punto di flesso è in $z = 0$ dove $\sigma(0) = 1/2$. L'output si interpreta come probabilità.

Tangente iperbolica $\tanh(z)$

$$\tanh(z) \triangleq \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

Comprime in $(-1, 1)$. È "centrata" in zero: $\tanh(0) = 0$. Vantaggio rispetto a sigmoid: avere output centrato in zero migliora la propagazione dei gradienti.

ReLU (Rectified Linear Unit)

$$\text{ReLU}(z) \triangleq \max(0, z) = \begin{cases} z & \text{se } z > 0 \\ 0 & \text{se } z \leq 0 \end{cases}$$

Semplice e computazionalmente efficiente. Il gradiente è 1 per $z > 0$ e 0 per $z < 0$ — non satura per valori positivi, evitando il vanishing gradient in quella regione. Problema: neuroni con $z \leq 0$ hanno gradiente zero e non si aggiornano ("neuroni morti").

GELU

$$\text{GELU}(z) \triangleq z \cdot \Phi(z)$$

dove $\Phi(z) = P(X \leq z)$ con $X \sim \mathcal{N}(0, 1)$ è la funzione di distribuzione cumulativa della gaussiana standard. In pratica si usa l'approssimazione:

$$\text{GELU}(z) \approx 0.5 z \left[1 + \tanh \left(\sqrt{\frac{2}{\pi}} (z + 0.044715 z^3) \right) \right]$$

Interpretazione: un input z viene "tenuto" con probabilità $\Phi(z)$ — se è grande e positivo, viene quasi sempre passato; se è negativo, viene quasi sempre azzerato, con una transizione morbida intorno a zero.

1.3 Derivate delle Funzioni di Attivazione — Passi Completi

Derivata della Sigmoid

Partiamo da $\sigma(z) = (1 + e^{-z})^{-1}$ e applichiamo la regola della catena.

1 Scriviamo $\sigma(z) = u^{-1}$ dove $u = 1 + e^{-z}$

Sostituzione per chiarezza

2
$$\frac{d\sigma}{dz} = \frac{d}{du}(u^{-1}) \cdot \frac{du}{dz} = -u^{-2} \cdot \frac{d}{dz}(1 + e^{-z})$$

Regola della catena

3
$$\frac{d}{dz}(1 + e^{-z}) = 0 + e^{-z} \cdot (-1) = -e^{-z}$$

Derivata di costante + derivata di e^{-z} (catena: e^u con $u = -z$, $u' = -1$)

4
$$\frac{d\sigma}{dz} = -(1 + e^{-z})^{-2} \cdot (-e^{-z}) = \frac{e^{-z}}{(1 + e^{-z})^2}$$

Sostituendo e semplificando il doppio segno meno

5
$$= \frac{1}{1 + e^{-z}} \cdot \frac{e^{-z}}{1 + e^{-z}} = \sigma(z) \cdot \frac{e^{-z}}{1 + e^{-z}}$$

Separazione della frazione

6
$$\frac{e^{-z}}{1 + e^{-z}} = \frac{(1 + e^{-z}) - 1}{1 + e^{-z}} = 1 - \frac{1}{1 + e^{-z}} = 1 - \sigma(z)$$

Aggiungiamo e sottraiamo 1 al numeratore, poi separiamo

$$\sigma'(z) = \sigma(z) (1 - \sigma(z))$$

Questa forma è elegante: per calcolare la derivata basta conoscere il valore $\sigma(z)$ già calcolato nel forward pass.

Derivata di tanh

Usiamo la regola del quoziente su $\tanh(z) = (e^z - e^{-z}) / (e^z + e^{-z})$. Poniamo $u = e^z - e^{-z}$ e $v = e^z + e^{-z}$:

$$1 \quad u' = e^z - (-e^{-z}) = e^z + e^{-z} = v$$

Derivata di e^z è e^z ; derivata di $-e^{-z}$ è $+e^{-z}$

$$2 \quad v' = e^z + e^{-z} = v$$

Uguale a u'

$$3 \quad \tanh'(z) = \frac{u'v - uv'}{v^2} = \frac{v \cdot v - u \cdot v}{v^2} = \frac{v^2 - u^2}{v^2}$$

Regola del quoziente: $(u/v)' = (u'v - uv')/v^2$

$$4 \quad = 1 - \frac{u^2}{v^2} = 1 - \left(\frac{u}{v}\right)^2 = 1 - \tanh^2(z)$$

Separazione e riconoscimento di tanh

$$\tanh'(z) = 1 - \tanh^2(z)$$

Derivata di ReLU (subgradiente)

ReLU non è differenziabile in $z = 0$ (il grafico ha uno spigolo). Per $z \neq 0$ la derivata ordinaria esiste:

$$\text{ReLU}'(z) = \begin{cases} 1 & \text{se } z > 0 \\ 0 & \text{se } z < 0 \\ \text{qualunque valore in } [0, 1] & \text{se } z = 0 \end{cases}$$

In pratica si usa $\text{ReLU}'(0) = 0$ per convenzione. La derivata vale 1 quando il neurone è "acceso" ($z > 0$) e 0 quando è "spento" — il gradiente passa invariato o viene bloccato.

PARTE 2

Multilayer Perceptron (MLP)

2.1 Architettura e Notazione Completa

Un MLP con L layer è una sequenza di trasformazioni. Prima di scrivere le formule, definiamo ogni simbolo che verrà usato:

Simbolo	Cosa rappresenta	Dimensione
L	Numero totale di layer (non conta l'input)	scalare intero
$n^{[0]}$	Numero di feature di input	scalare intero
$n^{[\ell]}$	Numero di neuroni nel layer ℓ (con $\ell = 1, \dots, L$)	scalare intero
$\mathbf{W}^{[\ell]}$	Matrice dei pesi del layer ℓ	$n^{[\ell]} \times n^{[\ell-1]}$
$\mathbf{b}^{[\ell]}$	Vettore dei bias del layer ℓ	$n^{[\ell]} \times 1$

$\mathbf{z}^{[\ell]}$	Pre-attivazione del layer ℓ (prima della funzione f)	$n^{[\ell]} \times 1$
$\mathbf{a}^{[\ell]}$	Attivazione del layer ℓ (dopo la funzione f)	$n^{[\ell]} \times 1$
$f^{[\ell]}$	Funzione di attivazione del layer ℓ (applicata componente per componente)	—
$\mathbf{a}^{[0]}$	Input della rete (alias $\mathbf{x}$)	$n^{[0]} \times 1$
$\hat{\mathbf{y}}$	Output predetto dalla rete (alias $\mathbf{a}^{[L]}$)	$n^{[L]} \times 1$
θ	Insieme di tutti i parametri: $\theta = \{\mathbf{W}^{[1]}, \mathbf{b}^{[1]}, \dots, \mathbf{W}^{[L]}, \mathbf{b}^{[L]}\}$	—

Conteggio dei parametri

In ogni layer ℓ , la matrice $\mathbf{W}^{[\ell]}$ ha $n^{[\ell]} \cdot n^{[\ell-1]}$ pesi, e il vettore $\mathbf{b}^{[\ell]}$ ha $n^{[\ell]}$ bias. Il totale è:

$$|\theta| = \sum_{\ell=1}^L \left(n^{[\ell]} \cdot n^{[\ell-1]} + n^{[\ell]} \right) = \sum_{\ell=1}^L n^{[\ell]} \left(n^{[\ell-1]} + 1 \right)$$

2.2 Forward Propagation

Il **forward pass** calcola l'output della rete dato un input $\mathbf{x}$. Si procede layer per layer, dal primo all'ultimo:

ALGORITMO — FORWARD PROPAGATION

Inizializzazione: $\mathbf{a}^{[0]} = \mathbf{x}$

Per ogni layer $\ell = 1, 2, \dots, L$:

$$\mathbf{z}^{[\ell]} = \mathbf{W}^{[\ell]} \mathbf{a}^{[\ell-1]} + \mathbf{b}^{[\ell]}$$

$$\mathbf{a}^{[\ell]} = f^{[\ell]}(\mathbf{z}^{[\ell]})$$

dove $f^{[\ell]}$ viene applicata componente per componente: $a_i^{[\ell]} = f^{[\ell]}(z_i^{[\ell]})$.

Output: $\hat{\mathbf{y}} = \mathbf{a}^{[L]}$

2.3 Esempio Numerico Completo — Forward Pass

Rete $2 \rightarrow 3 \rightarrow 1$ (2 input, 3 neuroni hidden con ReLU, 1 output con Sigmoid).

Input $\mathbf{x} = (1, 2)^T$, target $y = 1$.

Parametri del layer 1

$$\mathbf{W}^{[1]} = \begin{pmatrix} 0.1 & 0.4 \\ -0.2 & 0.3 \\ 0.5 & -0.1 \end{pmatrix} \in \mathbb{R}^{3 \times 2}, \quad \mathbf{b}^{[1]} = \begin{pmatrix} 0.1 \\ -0.2 \\ 0.3 \end{pmatrix} \in \mathbb{R}^3$$

Pre-attivazione del layer 1: $\mathbf{z}^{[1]} = \mathbf{W}^{[1]}\mathbf{x} + \mathbf{b}^{[1]}$

$$z_1^{[1]} = 0.1 \cdot 1 + 0.4 \cdot 2 + 0.1 = 0.1 + 0.8 + 0.1 = 1.0$$

$$z_2^{[1]} = -0.2 \cdot 1 + 0.3 \cdot 2 + (-0.2) = -0.2 + 0.6 - 0.2 = 0.2$$

$$z_3^{[1]} = 0.5 \cdot 1 + (-0.1) \cdot 2 + 0.3 = 0.5 - 0.2 + 0.3 = 0.6$$

$$\mathbf{z}^{[1]} = (1.0, 0.2, 0.6)^T$$

Attivazione ReLU: $\mathbf{a}^{[1]} = \text{ReLU}(\mathbf{z}^{[1]})$

Ogni componente: $\text{ReLU}(1.0) = 1.0$, $\text{ReLU}(0.2) = 0.2$, $\text{ReLU}(0.6) = 0.6$.

Tutte positive, nessun azzeramento.

$$\mathbf{a}^{[1]} = (1.0, 0.2, 0.6)^T$$

Parametri del layer 2

$$\mathbf{W}^{[2]} = \begin{pmatrix} 0.6 & -0.3 & 0.8 \end{pmatrix} \in \mathbb{R}^{1 \times 3}, \quad b^{[2]} = -0.5$$

Pre-attivazione del layer 2: $z^{[2]} = \mathbf{W}^{[2]} \mathbf{a}^{[1]} + b^{[2]}$

$$z^{[2]} = 0.6 \cdot 1.0 + (-0.3) \cdot 0.2 + 0.8 \cdot 0.6 + (-0.5) = 0.6 - 0.06 + 0.48 - 0.5 = 0.52$$

Attivazione Sigmoid: $\hat{y} = \sigma(z^{[2]})$

$$\hat{y} = \sigma(0.52) = \frac{1}{1 + e^{-0.52}}$$

Calcolo esplicito: $e^{-0.52} = e^{-0.52}$. Usiamo $e^{-0.5} \approx 0.6065$ e $e^{-0.02} \approx 0.9802$, quindi $e^{-0.52} \approx 0.5945$.

$$\hat{y} = \frac{1}{1 + 0.5945} = \frac{1}{1.5945} \approx 0.6271$$

La rete predice probabilità $\approx 62.7\%$ che l'esempio appartenga alla classe 1.

PARTE 3

3.1 Mean Squared Error (MSE)

Per la **regressione** con m esempi di training $(x^{(i)}, y^{(i)})$, il valore reale è $y^{(i)} \in \mathbb{R}$ e la predizione è $\hat{y}^{(i)} \in \mathbb{R}$:

$$\mathcal{L}_{\text{MSE}} = \frac{1}{m} \sum_{i=1}^m \left(y^{(i)} - \hat{y}^{(i)} \right)^2$$

Si eleva al quadrato per tre ragioni: (1) rende l'errore sempre non-negativo, (2) penalizza gli errori grandi più di quelli piccoli (quadraticamente), (3) è differenziabile ovunque.

Derivata rispetto a $\hat{y}^{(i)}$

Vogliamo sapere come cambia la loss al variare di una singola predizione $\hat{y}^{(i)}$.

Trattiamo gli altri termini come costanti:

- 1 Il termine i -esimo della somma è $(y^{(i)} - \hat{y}^{(i)})^2 \cdot \frac{1}{m}$

Solo questo dipende da $\hat{y}^{(i)}$

- 2 Sia $u = y^{(i)} - \hat{y}^{(i)}$. Allora $\partial u / \partial \hat{y}^{(i)} = -1$

La derivata di $-\hat{y}^{(i)}$ rispetto a $\hat{y}^{(i)}$ è -1

- 3 $\frac{\partial}{\partial \hat{y}^{(i)}} \left[\frac{1}{m} u^2 \right] = \frac{1}{m} \cdot 2u \cdot \frac{\partial u}{\partial \hat{y}^{(i)}} = \frac{2}{m} (y^{(i)} - \hat{y}^{(i)}) \cdot (-1)$

Regola della catena: derivata di u^2 è $2u$, poi moltiplico per $\partial u / \partial \hat{y}^{(i)}$

$$\frac{\partial \mathcal{L}_{\text{MSE}}}{\partial \hat{y}^{(i)}} = -\frac{2}{m} (y^{(i)} - \hat{y}^{(i)}) = \frac{2}{m} (\hat{y}^{(i)} - y^{(i)})$$

Interpretazione: se la predizione è maggiore del target ($\hat{y} > y$), il gradiente è positivo $\rightarrow$ dobbiamo diminuire $\hat{y}$. Se è minore, il gradiente è negativo $\rightarrow$ dobbiamo aumentarla.

3.2 Binary Cross-Entropy (BCE)

Per la **classificazione binaria** con $y \in \{0, 1\}$ e predizione $\hat{y} = \sigma(z) \in (0, 1)$:

$$\mathcal{L}_{\text{BCE}} = -\frac{1}{m} \sum_{i=1}^m \left[y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \right]$$

La formula ha questa forma perché:

- Se $y^{(i)} = 1$: la loss diventa $-\log \hat{y}^{(i)}$, che è 0 quando $\hat{y} = 1$ e $+\infty$ quando $\hat{y} \rightarrow 0$
- Se $y^{(i)} = 0$: la loss diventa $-\log(1 - \hat{y}^{(i)})$, che è 0 quando $\hat{y} = 0$ e $+\infty$ quando $\hat{y} \rightarrow 1$

Derivata della BCE rispetto al logit z (con output sigmoid)

Questo è uno dei risultati più importanti: la derivata $\partial \mathcal{L} / \partial z$ (e non $\partial \mathcal{L} / \partial \hat{y}$) ha una forma bellissima. Usiamo la regola della catena: $\partial \mathcal{L} / \partial z = (\partial \mathcal{L} / \partial \hat{y}) \cdot (\partial \hat{y} / \partial z)$.

- 1 Calcolo $\partial \mathcal{L} / \partial \hat{y}$ per un singolo esempio (ometto $1/m$):

$$\frac{\partial \mathcal{L}}{\partial \hat{y}} = -\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}} \cdot (-1) \cdot (-1) = -\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}}$$

Derivata di $y \log \hat{y}$: $y \cdot (1/\hat{y})$. Derivata di $(1 - y) \log(1 - \hat{y})$: $(1 - y) \cdot \frac{-1}{1-\hat{y}}$.

Tutto con segno meno davanti dalla definizione della loss.

- 2 Calcolo $\partial \hat{y} / \partial z$ dove $\hat{y} = \sigma(z)$:

$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1 - \sigma(z)) = \hat{y}(1 - \hat{y})$$

Derivata della sigmoid calcolata in 1.3

- 3 Moltiplico i due risultati:

$$\frac{\partial \mathcal{L}}{\partial z} = \left(-\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}} \right) \hat{y}(1 - \hat{y})$$

Regola della catena

- 4 Distribuisco il prodotto:

$$= -\frac{y}{\hat{y}} \cdot \hat{y}(1 - \hat{y}) + \frac{1-y}{1-\hat{y}} \cdot \hat{y}(1 - \hat{y}) = -y(1 - \hat{y}) + (1 - y)\hat{y}$$

Semplificazione delle frazioni: $\hat{y}/\hat{y} = 1$ e $(1 - \hat{y})/(1 - \hat{y}) = 1$

- 5 Espando e raccolgo:

$$= -y + y\hat{y} + \hat{y} - y\hat{y} = \hat{y} - y$$

$y\hat{y}$ e $-y\hat{y}$ si cancellano

$$\boxed{\frac{\partial \mathcal{L}_{\text{BCE}}}{\partial z} = \hat{y} - y}$$

Quando $y = 1$ e $\hat{y} = 0.6$: gradiente = -0.4 (spingi z verso l'alto per aumentare $\hat{y}$). Quando $y = 0$ e $\hat{y} = 0.3$: gradiente = $+0.3$ (spingi z verso il basso per diminuire $\hat{y}$).

3.3 Softmax e Categorical Cross-Entropy

Softmax: trasformare logit in probabilità

Per la classificazione con K classi, l'ultimo layer produce K logit $z_1, \dots, z_K \in \mathbb{R}$.

La softmax li converte in probabilità:

$$\hat{y}_k = \text{softmax}(\mathbf{z})_k \triangleq \frac{e^{z_k}}{\sum_{j=1}^K e^{z_j}}, \quad k = 1, \dots, K$$

Si verifica facilmente che $\hat{y}_k > 0$ per ogni k (l'esponenziale è sempre positivo) e $\sum_k \hat{y}_k = \sum_k e^{z_k} / \sum_j e^{z_j} = 1$. Quindi è effettivamente una distribuzione di probabilità.

Jacobiana della Softmax

La softmax mappa $\mathbf{z} \in \mathbb{R}^K \rightarrow \hat{\mathbf{y}} \in \mathbb{R}^K$. La sua derivata è una **matrice** $K \times K$ (la Jacobiana). Per i due casi $k = j$ e $k \neq j$:

Caso $k = j$: $\partial \hat{y}_k / \partial z_k$

1 Scriviamo $\hat{y}_k = e^{z_k} / S$ dove $S = \sum_j e^{z_j}$

Abbreviazione

2
$$\frac{\partial \hat{y}_k}{\partial z_k} = \frac{\frac{\partial}{\partial z_k}(e^{z_k}) \cdot S - e^{z_k} \cdot \frac{\partial S}{\partial z_k}}{S^2}$$

Regola del quoziente

3
$$\frac{\partial}{\partial z_k}(e^{z_k}) = e^{z_k} \text{ e } \frac{\partial S}{\partial z_k} = e^{z_k} \text{ (solo il termine } k \text{ dipende da } z_k)$$

Gli altri termini della somma non dipendono da z_k

4
$$= \frac{e^{z_k} S - e^{z_k} e^{z_k}}{S^2} = \frac{e^{z_k}}{S} - \frac{e^{z_k}}{S} \cdot \frac{e^{z_k}}{S} = \hat{y}_k - \hat{y}_k^2 = \hat{y}_k(1 - \hat{y}_k)$$

Separazione e riconoscimento di $\hat{y}_k = e^{z_k} / S$

Caso $k \neq j$: $\partial \hat{y}_k / \partial z_j$

$$1 \quad \frac{\partial \hat{y}_k}{\partial z_j} = \frac{\frac{\partial}{\partial z_j}(e^{z_k}) \cdot S - e^{z_k} \cdot \frac{\partial S}{\partial z_j}}{S^2}$$

Quoziente di nuovo

$$2 \quad \frac{\partial}{\partial z_j}(e^{z_k}) = 0 \text{ perché } k \neq j \text{ (il termine } k \text{ non dipende da } z_j)$$

Costante rispetto a z_j

$$3 \quad = \frac{0 - e^{z_k} \cdot e^{z_j}}{S^2} = -\frac{e^{z_k}}{S} \cdot \frac{e^{z_j}}{S} = -\hat{y}_k \hat{y}_j$$

Usando il **delta di Kronecker** δ_{kj} (che vale 1 se $k = j$ e 0 se $k \neq j$), si unificano i due casi:

$$\frac{\partial \hat{y}_k}{\partial z_j} = \hat{y}_k(\delta_{kj} - \hat{y}_j)$$

Categorical Cross-Entropy e il suo gradiente

Con K classi e vettore one-hot $\mathbf{y}$ (tutti zero tranne un 1 nella posizione della classe corretta c):

$$\mathcal{L}_{\text{CCE}} = - \sum_{k=1}^K y_k \log \hat{y}_k = -\log \hat{y}_c$$

Gradiente rispetto al logit z_j (usando regola della catena e la Jacobiana appena calcolata):

$$1 \quad \frac{\partial \mathcal{L}}{\partial z_j} = \sum_{k=1}^K \frac{\partial \mathcal{L}}{\partial \hat{y}_k} \cdot \frac{\partial \hat{y}_k}{\partial z_j}$$

Regola della catena sommando su tutti i k (ogni $\hat{y}_k$ dipende da z_j)

$$2 \quad \frac{\partial \mathcal{L}}{\partial \hat{y}_k} = -\frac{y_k}{\hat{y}_k}$$

Derivata di $-y_k \log \hat{y}_k$ rispetto a $\hat{y}_k$: $-y_k \cdot (1/\hat{y}_k)$

$$3 \quad \frac{\partial \mathcal{L}}{\partial z_j} = \sum_{k=1}^K \left(-\frac{y_k}{\hat{y}_k} \right) \hat{y}_k (\delta_{kj} - \hat{y}_j) = -\sum_{k=1}^K y_k (\delta_{kj} - \hat{y}_j)$$

Sostituzione e cancellazione di $\hat{y}_k$

$$4 \quad = -\sum_{k=1}^K y_k \delta_{kj} + \hat{y}_j \sum_{k=1}^K y_k = -y_j + \hat{y}_j \cdot 1$$

$\sum_k y_k \delta_{kj} = y_j$ (solo il termine $k = j$ sopravvive); $\sum_k y_k = 1$ (one-hot)

$$\boxed{\frac{\partial \mathcal{L}_{\text{CCE}}}{\partial z_j} = \hat{y}_j - y_j}$$

Stesso risultato del caso binario: la derivata della CCE+Softmax rispetto ai logit è semplicemente la differenza tra predizione e target. Questo è valido per ogni $j = 1, \dots, K$.

PARTE 4

Backpropagation

4.1 La Regola della Catena per Grafi Computazionali

Una rete neurale è un grafo di operazioni composte. Per trovare $\partial\mathcal{L}/\partial w$ per qualsiasi peso w , si usa la regola della catena attraverso tutte le operazioni intermedie.

Se la computazione è $w \rightarrow z \rightarrow a \rightarrow \mathcal{L}$, allora:

$$\frac{\partial\mathcal{L}}{\partial w} = \frac{\partial\mathcal{L}}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial w}$$

Ogni fattore è il "gradiente locale" di un'operazione. Si moltiplicano partendo dall'output (la loss) verso l'input (i pesi) — da qui il nome **backpropagation**.

4.2 Algoritmo Completo

Definizione del gradiente di layer

Introduciamo la notazione $\delta^{[\ell]}$ per il gradiente della loss rispetto alla *pre-attivazione* del layer ℓ :

$$\delta^{[\ell]} \triangleq \frac{\partial\mathcal{L}}{\partial \mathbf{z}^{[\ell]}} \in \mathbb{R}^{n^{[\ell]}}$$

Una volta che conosciamo $\delta^{[\ell]}$, i gradienti dei parametri di quel layer si calcolano facilmente. Derivando $\mathbf{z}^{[\ell]} = \mathbf{W}^{[\ell]}\mathbf{a}^{[\ell-1]} + \mathbf{b}^{[\ell]}$ rispetto a $\mathbf{W}^{[\ell]}$ e $\mathbf{b}^{[\ell]}$:

$$\begin{aligned} \frac{\partial\mathcal{L}}{\partial \mathbf{W}^{[\ell]}} &= \delta^{[\ell]} \left(\mathbf{a}^{[\ell-1]} \right)^T \in \mathbb{R}^{n^{[\ell]} \times n^{[\ell-1]}} \\ \frac{\partial\mathcal{L}}{\partial \mathbf{b}^{[\ell]}} &= \delta^{[\ell]} \in \mathbb{R}^{n^{[\ell]}} \end{aligned}$$

La formula per $\mathbf{W}$ è un **prodotto esterno**: moltiplicando il vettore colonna $\delta^{[\ell]}$ per il vettore riga $(\mathbf{a}^{[\ell-1]})^T$ si ottiene una matrice della giusta dimensione.

Derivazione del prodotto esterno

Consideriamo un singolo peso $w_{ij}^{[\ell]}$ (riga i , colonna j di $\mathbf{W}^{[\ell]}$). Dato che $z_i^{[\ell]} = \sum_j w_{ij}^{[\ell]} a_j^{[\ell-1]} + b_i^{[\ell]}$:

$$\frac{\partial z_i^{[\ell]}}{\partial w_{ij}^{[\ell]}} = a_j^{[\ell-1]}$$

Quindi, per la regola della catena:

$\partial \mathcal{L} / \partial w_{ij}^{[\ell]} = (\partial \mathcal{L} / \partial z_i^{[\ell]}) \cdot (\partial z_i^{[\ell]} / \partial w_{ij}^{[\ell]}) = \delta_i^{[\ell]} \cdot a_j^{[\ell-1]}$. Scrivendo tutti gli elementi in forma matriciale si ottiene esattamente il prodotto esterno.

Come propagare δ all'indietro

Come si calcola $\delta^{[\ell]}$ a partire da $\delta^{[\ell+1]}$? Usando la regola della catena tra i layer:

REGOLA DI RICORRENZA DEL BACKPROPAGATION

Step 1 — Propagare il gradiente dalla pre-attivazione $\mathbf{z}^{[\ell+1]}$ all'attivazione $\mathbf{a}^{[\ell]}$:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[\ell]}} = \left(\mathbf{W}^{[\ell+1]} \right)^T \delta^{[\ell+1]}$$

Step 2 — Propagare da $\mathbf{a}^{[\ell]}$ a $\mathbf{z}^{[\ell]}$ (moltiplicare per la derivata dell'attivazione):

$$\delta^{[\ell]} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[\ell]}} \odot f'^{[\ell]}(\mathbf{z}^{[\ell]})$$

Il simbolo $\odot$ è il prodotto di Hadamard (componente per componente): la derivata di f viene applicata a ogni componente di $\mathbf{z}^{[\ell]}$ separatamente.

Perché $(\mathbf{W}^{[\ell+1]})^T$?

La connessione tra layer ℓ e $\ell + 1$ è $z_i^{[\ell+1]} = \sum_j w_{ij}^{[\ell+1]} a_j^{[\ell]} + b_i^{[\ell+1]}$. Quindi $\partial z_i^{[\ell+1]} / \partial a_j^{[\ell]} = w_{ij}^{[\ell+1]}$. Applicando la regola della catena:

$$\frac{\partial \mathcal{L}}{\partial a_j^{[\ell]}} = \sum_{i=1}^{n^{[\ell+1]}} \frac{\partial \mathcal{L}}{\partial z_i^{[\ell+1]}} \cdot w_{ij}^{[\ell+1]} = \sum_i \delta_i^{[\ell+1]} w_{ij}^{[\ell+1]}$$

Scrivendo in forma vettoriale: $\partial \mathcal{L} / \partial \mathbf{a}^{[\ell]} = (\mathbf{W}^{[\ell+1]})^T \boldsymbol{\delta}^{[\ell+1]}$. La trasposta appare perché stiamo "invertendo" la moltiplicazione matrice-vettore.

4.3 Esempio Numerico Completo

Continuiamo la rete dell'esempio 2.3 (rete $2 \rightarrow 3 \rightarrow 1$). I valori del forward pass erano:

$$\mathbf{x} = (1, 2)^T, \mathbf{a}^{[1]} = (1.0, 0.2, 0.6)^T, \hat{y} = 0.6271, y = 1$$

Loss e gradiente iniziale (layer output)

$\mathcal{L} = -\log(0.6271) \approx 0.4665$. Il gradiente rispetto al logit $z^{[2]}$ è:

$$\delta^{[2]} = \hat{y} - y = 0.6271 - 1 = -0.3729$$

Gradienti dei parametri del layer 2

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{[2]}} = \delta^{[2]} \cdot (\mathbf{a}^{[1]})^T = (-0.3729) \cdot (1.0, 0.2, 0.6) = (-0.3729, -0.0746, -0.2237)$$

$$\frac{\partial \mathcal{L}}{\partial b^{[2]}} = \delta^{[2]} = -0.3729$$

Propagazione al layer 1 — Step 1

$$\frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[1]}} = (\mathbf{W}^{[2]})^T \delta^{[2]} = \begin{pmatrix} 0.6 \\ -0.3 \\ 0.8 \end{pmatrix} \cdot (-0.3729) = \begin{pmatrix} -0.2237 \\ 0.1119 \\ -0.2983 \end{pmatrix}$$

Propagazione al layer 1 — Step 2

Derivata di ReLU: $\text{ReLU}'(z_i) = 1$ se $z_i > 0$, else 0. I valori

$\mathbf{z}^{[1]} = (1.0, 0.2, 0.6)^T$ sono tutti > 0 , quindi $f'^{[1]}(\mathbf{z}^{[1]}) = (1, 1, 1)^T$.

$$\delta^{[1]} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[1]}} \odot f'^{[1]}(\mathbf{z}^{[1]}) = \begin{pmatrix} -0.2237 \\ 0.1119 \\ -0.2983 \end{pmatrix} \odot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} -0.2237 \\ 0.1119 \\ -0.2983 \end{pmatrix}$$

Gradienti dei parametri del layer 1

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{[1]}} = \delta^{[1]} (\mathbf{x})^T = \begin{pmatrix} -0.2237 \\ 0.1119 \\ -0.2983 \end{pmatrix} (1, 2) = \begin{pmatrix} -0.2237 & -0.4474 \\ 0.1119 & 0.2238 \\ -0.2983 & -0.5966 \end{pmatrix}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}^{[1]}} = \delta^{[1]} = (-0.2237, 0.1119, -0.2983)^T$$

Abbiamo ora il gradiente rispetto a ogni parametro della rete. L'aggiornamento con learning rate $\alpha = 0.1$: $\mathbf{W} \leftarrow \mathbf{W} - 0.1 \cdot \nabla_{\mathbf{W}} \mathcal{L}$.

PARTE 5

Ottimizzatori

5.1 Gradient Descent e Varianti

Gradient descent base

Il principio: aggiornare i parametri nella direzione di discesa più ripida della loss.

Per ogni parametro θ (può essere un singolo peso o il vettore completo):

$$\theta \leftarrow \theta - \alpha \frac{\partial \mathcal{L}}{\partial \theta}$$

Il simbolo $\leftarrow$ indica un aggiornamento: il nuovo valore di θ è il vecchio meno α volte il gradiente. $\alpha > 0$ è il **learning rate** (tasso di apprendimento) — controlla l'ampiezza del passo.

Scelta del learning rate

Per una loss quadratica semplice $\mathcal{L}(\theta) = \frac{1}{2}\theta^2$ (con gradiente $\nabla \mathcal{L} = \theta$),

l'aggiornamento è $\theta_{t+1} = \theta_t - \alpha\theta_t = (1 - \alpha)\theta_t$. Dopo T passi:

$$\theta_T = (1 - \alpha)^T \theta_0.$$

- $0 < \alpha < 1$: $|1 - \alpha| < 1$, quindi $|\theta_T| \rightarrow 0$ — convergenza

- $\alpha = 1$: $\theta_1 = 0$ in un solo passo — ottimale per questa loss
- $1 < \alpha < 2$: oscillazioni che però convergono
- $\alpha \geq 2$: $|1 - \alpha| \geq 1$, la loss diverge

Mini-batch SGD

Invece di calcolare il gradiente sull'intero dataset (m esempi), si usa un sottinsieme casuale di B esempi detto **mini-batch** $\mathcal{B}$:

$$\theta \leftarrow \theta - \alpha \frac{1}{B} \sum_{i \in \mathcal{B}} \frac{\partial \mathcal{L}^{(i)}}{\partial \theta}$$

Il gradiente del mini-batch è una stima rumorosa del gradiente vero. Il rumore ha un effetto regolarizzante: aiuta a uscire dai minimi locali stretti.

5.2 Adam — Ogni Passo Spiegato

Motivazione: learning rate adattivo

Il problema del gradient descent: lo stesso learning rate α per tutti i parametri è inefficiente. Alcuni parametri hanno gradienti molto variabili (servono passi piccoli), altri consistenti e piccoli (servono passi grandi). Adam risolve questo adattando il learning rate per ogni parametro individualmente.

Due statistiche per ogni parametro

Adam mantiene due medie mobili esponenziali del gradiente $g_t = \partial \mathcal{L} / \partial \theta$ al passo t :

Primo momento (media mobile del gradiente, stima la direzione media):

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

Con $\beta_1 = 0.9$: ogni nuovo m_t è il 90% del vecchio m_{t-1} più il 10% del gradiente corrente g_t . Questo dà "inerzia" — la direzione dell'aggiornamento non cambia bruscamente.

Secondo momento (media mobile del gradiente al quadrato, stima la magnitudine):

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

Con $\beta_2 = 0.999$: media molto lenta. v_t accumula la magnitudine storica del gradiente. Se un parametro ha spesso gradienti grandi, v_t sarà grande.

Correzione del bias — perché è necessaria

Al passo $t = 1$, partendo da $m_0 = 0$ e $v_0 = 0$:

$$m_1 = \beta_1 \cdot 0 + (1 - \beta_1) g_1 = (1 - \beta_1) g_1$$

Con $\beta_1 = 0.9$: $m_1 = 0.1 g_1$. Questo *sottostima* la vera media (che dovrebbe essere g_1) di un fattore $1 - \beta_1 = 0.1$. La correzione divide per $(1 - \beta_1^t)$:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

Al passo $t = 1$: $\hat{m}_1 = m_1 / (1 - 0.9) = 0.1g_1 / 0.1 = g_1$ ✓. Al passo $t = 100$: $\beta_1^{100} = 0.9^{100} \approx 2.7 \times 10^{-5} \approx 0$, quindi $(1 - \beta_1^t) \approx 1$ e la correzione diventa trascurabile.

L'aggiornamento finale

ALGORITMO ADAM — PER OGNI PARAMETRO, PER OGNI PASSO T

$$\begin{aligned}
 g_t &= \left. \frac{\partial \mathcal{L}}{\partial \theta} \right|_t \\
 m_t &= \beta_1 m_{t-1} + (1 - \beta_1) g_t \\
 v_t &= \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \\
 \hat{m}_t &= \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t} \\
 \theta_t &= \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon}
 \end{aligned}$$

Valori tipici: $\alpha = 3 \times 10^{-4}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 10^{-8}$.

Il termine $\sqrt{\hat{v}_t}$ al denominatore scala il passo di ogni parametro inversamente alla magnitudine storica del suo gradiente. Il $+\varepsilon$ evita la divisione per zero.

PARTE 6

Regolarizzazione

6.1 Regolarizzazione L1 e L2

Il problema dell'overfitting

Una rete con molti parametri può imparare a memorizzare il training set, dando errore zero su di esso ma funzionando male su nuovi dati. La regolarizzazione aggiunge alla loss un termine di penalità che scoraggia pesi troppo grandi:

$$\mathcal{L}_{\text{reg}}(\theta) = \mathcal{L}(\theta) + \lambda \Omega(\theta)$$

dove $\lambda > 0$ è l'**iperparametro di regolarizzazione** (controlla quanto pesiamo la penalità) e $\Omega(\theta)$ è il termine di penalità.

L2 — Weight Decay

$$\Omega_2(\theta) = \frac{1}{2} \sum_k \theta_k^2 = \frac{1}{2} \|\theta\|_2^2$$

La derivata rispetto a θ_k è semplicemente $\partial\Omega_2/\partial\theta_k = \theta_k$. L'aggiornamento diventa:

$$\theta_k \leftarrow \theta_k - \alpha \left(\frac{\partial \mathcal{L}}{\partial \theta_k} + \lambda \theta_k \right) = (1 - \alpha \lambda) \theta_k - \alpha \frac{\partial \mathcal{L}}{\partial \theta_k}$$

Il fattore $(1 - \alpha \lambda) < 1$ contrae θ_k verso zero ad ogni passo — da cui "weight decay". I pesi grandi vengono penalizzati più di quelli piccoli.

L1 — Lasso

$$\Omega_1(\theta) = \sum_k |\theta_k| = \|\theta\|_1$$

La derivata del valore assoluto è il segno: $\partial|\theta_k|/\partial\theta_k = \text{sign}(\theta_k)$. L'aggiornamento è:

$$\theta_k \leftarrow \theta_k - \alpha \frac{\partial \mathcal{L}}{\partial \theta_k} - \alpha \lambda \text{sign}(\theta_k)$$

L1 spinge i pesi verso zero di un passo fisso $\alpha\lambda$ (indipendente dalla grandezza del peso), portando molti pesi esattamente a zero — produce **sparsità**.

Connessione con l'interpretazione Bayesiana

Minimizzare $\mathcal{L} + \frac{\lambda}{2} \|\theta\|^2$ equivale a massimizzare il logaritmo della posterior $\log p(\theta|\mathcal{D}) = \log p(\mathcal{D}|\theta) + \log p(\theta) + \text{cost.}$ con prior gaussiano $p(\theta) = \mathcal{N}(0, 1/\lambda)$. L1 corrisponde a un prior di Laplace $p(\theta_k) \propto e^{-\lambda|\theta_k|}$.

6.2 Dropout

Durante ogni forward pass di training, ogni neurone viene azzerato indipendentemente con probabilità p (tipicamente $p \in [0.2, 0.5]$):

$$\tilde{a}_j^{[\ell]} = m_j^{[\ell]} \cdot a_j^{[\ell]}, \quad m_j^{[\ell]} \sim \text{Bernoulli}(1 - p)$$

$m_j^{[\ell]} = 1$ con probabilità $(1 - p)$ e 0 con probabilità p . Ogni batch usa maschere diverse, estratte casualmente.

Inverted dropout: per mantenere lo stesso valore atteso dell'attivazione, si scala per $1/(1 - p)$:

$$\tilde{a}_j^{[\ell]} = \frac{m_j^{[\ell]}}{1 - p} \cdot a_j^{[\ell]}$$

Così $\mathbb{E}[\tilde{a}_j] = \frac{1}{1-p} \cdot (1 - p) \cdot a_j + 0 \cdot p = a_j$ — stessa aspettativa. Durante l'inferenza si usa la rete completa senza dropout e senza scalatura.

6.3 Batch Normalization — Ogni Passo

BN normalizza le attivazioni di ogni layer su un mini-batch, rendendo l'addestramento più stabile.

Dato un mini-batch di m pre-attivazioni per una singola feature j :

$$\{z_j^{(1)}, z_j^{(2)}, \dots, z_j^{(m)}\}:$$

BATCH NORMALIZATION — 4 PASSI

Passo 1 — Media del mini-batch:

$$\mu_j = \frac{1}{m} \sum_{i=1}^m z_j^{(i)}$$

Passo 2 — Varianza del mini-batch:

$$\sigma_j^2 = \frac{1}{m} \sum_{i=1}^m \left(z_j^{(i)} - \mu_j \right)^2$$

Passo 3 — Normalizzazione (media 0, varianza 1):

$$\hat{z}_j^{(i)} = \frac{z_j^{(i)} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Il termine ε (tipicamente 10^{-5}) evita la divisione per zero se $\sigma_j^2 = 0$.

Passo 4 — Riscaldamento con parametri appresi γ_j e β_j :

$$\text{BN}(z_j^{(i)}) = \gamma_j \hat{z}_j^{(i)} + \beta_j$$

I parametri γ_j e β_j permettono alla rete di "disimparare" la normalizzazione se è controproducente: se $\gamma_j = \sigma_j$ e $\beta_j = \mu_j$, BN diventa l'identità.

PARTE 7

Reti Convoluzionali (CNN)

7.1 La Convoluzione Discreta — Ogni Moltiplicazione Esplicita

Cos'è un'immagine come tensore

Un'immagine a colori di dimensioni $H \times W$ con C canali (es. RGB: $C = 3$) è un tensore $\mathbf{X} \in \mathbb{R}^{C \times H \times W}$. Ogni elemento $\mathbf{X}[c, i, j] \in [0, 255]$ è l'intensità del canale c nel pixel alla riga i , colonna j .

Convoluzione 2D con un singolo canale

Per un'immagine a singolo canale $\mathbf{X} \in \mathbb{R}^{H \times W}$ e un filtro (kernel) $\mathbf{W} \in \mathbb{R}^{k \times k}$, la feature map di output $\mathbf{Z} \in \mathbb{R}^{H_{\text{out}} \times W_{\text{out}}}$ è:

$$\mathbf{Z}[i, j] = \sum_{p=0}^{k-1} \sum_{q=0}^{k-1} \mathbf{X}[i + p, j + q] \cdot \mathbf{W}[p, q] + b$$

Per ogni posizione (i, j) : si estrae una "finestra" di dimensione $k \times k$ da $\mathbf{X}$ centrata intorno a (i, j) , si moltiplica elemento per elemento con il filtro $\mathbf{W}$, e si sommano tutti i k^2 prodotti.

Dimensioni dell'output

Con padding P (righe/colonne di zeri aggiunte ai bordi) e stride S (quanti pixel si salta tra un posizionamento e il successivo del filtro):

$$H_{\text{out}} = \left\lfloor \frac{H + 2P - k}{S} \right\rfloor + 1, \quad W_{\text{out}} = \left\lfloor \frac{W + 2P - k}{S} \right\rfloor + 1$$

Il simbolo $\lfloor \cdot \rfloor$ è la parte intera inferiore (floor). Esempio: $H = 5, k = 3, P = 0, S = 1$: $H_{\text{out}} = \lfloor (5 + 0 - 3)/1 \rfloor + 1 = 2 + 1 = 3$.

ESEMPIO NUMERICO — OGNI PRODOTTO ESPlicito

Input 4×4 , filtro 2×2 , $P = 0, S = 1$. Output: 3×3 .

$$\mathbf{X} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{pmatrix}, \quad \mathbf{W} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \quad b = 0$$

Calcolo di $\mathbf{Z}[0, 0]$ (finestra in alto a sinistra):

$$\mathbf{Z}[0, 0] = 1 \cdot 1 + 2 \cdot 0 + 5 \cdot 0 + 6 \cdot (-1) = 1 + 0 + 0 - 6 = -5$$

Calcolo di $\mathbf{Z}[0, 1]$ (finestra spostata di 1 a destra):

$$\mathbf{Z}[0, 1] = 2 \cdot 1 + 3 \cdot 0 + 6 \cdot 0 + 7 \cdot (-1) = 2 + 0 + 0 - 7 = -5$$

Calcolo di $\mathbf{Z}[1, 0]$ (finestra spostata di 1 in basso):

$$\mathbf{Z}[1, 0] = 5 \cdot 1 + 6 \cdot 0 + 9 \cdot 0 + 10 \cdot (-1) = 5 - 10 = -5$$

Il filtro $\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$ calcola la differenza tra il pixel in alto a sinistra e quello in basso a destra di ogni 2×2 blocco.

Convoluzione con canali multipli e filtri multipli

Input con C_{in} canali, F filtri (che producono F feature maps). Il tensore dei pesi è $\mathbf{W} \in \mathbb{R}^{F \times C_{\text{in}} \times k \times k}$, bias $\mathbf{b} \in \mathbb{R}^F$. La f -esima feature map è:

$$\mathbf{Z}^{(f)}[i, j] = \sum_{c=1}^{C_{\text{in}}} \sum_{p=0}^{k-1} \sum_{q=0}^{k-1} \mathbf{X}^{(c)}[i+p, j+q] \cdot \mathbf{W}^{(f,c)}[p, q] + b_f$$

Ogni filtro f "guarda" tutti i canali input, sommandone i contributi. Il risultato è un tensore $\mathbf{Z} \in \mathbb{R}^{F \times H_{\text{out}} \times W_{\text{out}}}$.

Backpropagation nella convoluzione

Dato il gradiente $\partial \mathcal{L} / \partial \mathbf{Z}$ che arriva dallo strato successivo:

Gradiente rispetto al filtro — per ogni peso $\mathbf{W}[p, q]$, sommo il contributo di tutte le posizioni (i, j) dove il filtro è stato applicato:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}[p, q]} = \sum_{i, j} \frac{\partial \mathcal{L}}{\partial \mathbf{Z}[i, j]} \cdot \mathbf{X}[i+p, j+q]$$

Gradiente rispetto all'input — per ogni pixel $\mathbf{X}[r, s]$, sommo i contributi di tutte le posizioni del filtro che "vedono" quel pixel:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{X}[r, s]} = \sum_{p, q} \frac{\partial \mathcal{L}}{\partial \mathbf{Z}[r - p, s - q]} \cdot \mathbf{W}[p, q]$$

Questa operazione è la **convoluzione trasposta** (o deconvoluzione): stessa struttura della convoluzione ma con il filtro capovolto e padding "pieno".

7.2 Pooling e il suo Backpropagation

Max Pooling

Suddivide la feature map in finestre $p \times p$ e prende il massimo in ciascuna:

$$\mathbf{Z}_{\text{pool}}[i, j] = \max_{0 \leq r, c < p} \mathbf{X}[i \cdot S + r, j \cdot S + c]$$

Backward: il gradiente fluisce solo verso l'elemento che aveva il massimo (indice (r^*, c^*)). Gli altri ricevono gradiente zero:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{X}[i \cdot S + r, j \cdot S + c]} = \begin{cases} \frac{\partial \mathcal{L}}{\partial \mathbf{Z}_{\text{pool}}[i, j]} & \text{se } (r, c) = (r^*, c^*) \\ 0 & \text{altrimenti} \end{cases}$$

Average Pooling

Prende la media: $\mathbf{Z}_{\text{avg}}[i, j] = \frac{1}{p^2} \sum_{r, c} \mathbf{X}[i \cdot S + r, j \cdot S + c]$. **Backward:** il gradiente viene distribuito uniformemente su tutti i p^2 elementi della finestra:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{X}[i \cdot S + r, j \cdot S + c]} = \frac{1}{p^2} \frac{\partial \mathcal{L}}{\partial \mathbf{Z}_{\text{avg}}[i, j]}$$

7.3 Residual Connections — Perché Risolvono il Vanishing Gradient

In una rete profonda senza residual connections, il gradiente attraversa ogni layer moltiplicandosi per $\partial \mathbf{a}^{[\ell]} / \partial \mathbf{a}^{[\ell-1]}$. Se questo fattore è < 1 in norma, dopo L layer il gradiente si riduce di un fattore $(\approx c)^L \rightarrow 0$.

Il blocco residuale aggiunge un **shortcut path**:

$$\mathbf{a}^{[\ell+1]} = f^{[\ell+1]}(\mathbf{W}^{[\ell+1]} \mathbf{a}^{[\ell]} + \mathbf{b}^{[\ell+1]}) + \mathbf{a}^{[\ell]}$$

Il gradiente rispetto a $\mathbf{a}^{[\ell]}$ è ora:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[\ell]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[\ell+1]}} \left(\frac{\partial \mathcal{F}}{\partial \mathbf{a}^{[\ell]}} + \mathbf{I} \right)$$

Il termine $+\mathbf{I}$ (matrice identità) garantisce che il gradiente raggiunga sempre i layer profondi. Anche se $\partial \mathcal{F} / \partial \mathbf{a}^{[\ell]} \rightarrow 0$, il gradiente non sparisce: almeno la componente $\partial \mathcal{L} / \partial \mathbf{a}^{[\ell+1]}$ viene trasmessa intatta.

8.1 RNN Base

Una sequenza è una lista ordinata di vettori: $(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T)$ con $\mathbf{x}_t \in \mathbb{R}^n$. Il problema è che una rete feedforward non può gestire sequenze di lunghezza variabile, né sfruttare l'ordine temporale.

Una RNN mantiene uno **stato nascosto** $\mathbf{h}_t \in \mathbb{R}^d$ che viene aggiornato ad ogni passo:

EQUAZIONI RNN (ELMAN)

Inizializzazione: $\mathbf{h}_0 = \mathbf{0}$ (vettore di zeri)

Per $t = 1, 2, \dots, T$:

$$\mathbf{z}_t = \mathbf{W}_{hh} \mathbf{h}_{t-1} + \mathbf{W}_{xh} \mathbf{x}_t + \mathbf{b}_h$$

$$\mathbf{h}_t = \tanh(\mathbf{z}_t)$$

$$\mathbf{y}_t = \mathbf{W}_{hy} \mathbf{h}_t + \mathbf{b}_y$$

Parametri: $\mathbf{W}_{hh} \in \mathbb{R}^{d \times d}$ (da hidden a hidden), $\mathbf{W}_{xh} \in \mathbb{R}^{d \times n}$ (da input a hidden), $\mathbf{W}_{hy} \in \mathbb{R}^{m \times d}$ (da hidden a output). Cruciale: gli stessi pesi vengono usati ad ogni passo t — questo si chiama **weight sharing through time**.

8.2 BPTT e Vanishing Gradient — Derivazione Passo per Passo

La loss di sequenza

$$\mathcal{L} = \sum_{t=1}^T \mathcal{L}_t, \quad \text{es. } \mathcal{L}_t = -\log P(x_{t+1}|x_1, \dots, x_t)$$

Gradiente di $\mathcal{L}_T$ rispetto a $\mathbf{W}_{hh}$

Usiamo la regola della catena. $\mathcal{L}_T$ dipende da $\mathbf{h}_T$, che dipende da $\mathbf{h}_{T-1}$, che dipende da $\mathbf{h}_{T-2}$, ..., che dipende da $\mathbf{W}_{hh}$. Quindi:

$$\frac{\partial \mathcal{L}_T}{\partial \mathbf{W}_{hh}} = \sum_{k=1}^T \frac{\partial \mathcal{L}_T}{\partial \mathbf{h}_T} \cdot \frac{\partial \mathbf{h}_T}{\partial \mathbf{h}_k} \cdot \frac{\partial \mathbf{h}_k}{\partial \mathbf{W}_{hh}}$$

Il termine critico è $\partial \mathbf{h}_T / \partial \mathbf{h}_k$, che per la regola della catena è:

$$\frac{\partial \mathbf{h}_T}{\partial \mathbf{h}_k} = \prod_{j=k+1}^T \frac{\partial \mathbf{h}_j}{\partial \mathbf{h}_{j-1}}$$

Ogni fattore è la Jacobiana di $\mathbf{h}_j$ rispetto a $\mathbf{h}_{j-1}$. Da $\mathbf{h}_j = \tanh(\mathbf{W}_{hh} \mathbf{h}_{j-1} + \dots)$:

$$\frac{\partial \mathbf{h}_j}{\partial \mathbf{h}_{j-1}} = \text{diag}(\tanh'(\mathbf{z}_j)) \cdot \mathbf{W}_{hh} = \text{diag}(1 - \mathbf{h}_j^2) \cdot \mathbf{W}_{hh}$$

dove $\text{diag}(\mathbf{v})$ è la matrice diagonale con il vettore $\mathbf{v}$ sulla diagonale. Il prodotto per $T - k$ fattori dà:

$$\left\| \frac{\partial \mathbf{h}_T}{\partial \mathbf{h}_k} \right\| \leq \prod_{j=k+1}^T \left\| \text{diag}(1 - \mathbf{h}_j^2) \right\| \cdot \|\mathbf{W}_{hh}\| \leq (\gamma \cdot \|\mathbf{W}_{hh}\|)^{T-k}$$

dove $\gamma = \max_j(1 - h_{j,\max}^2) \leq 1$ (poiché $\tanh' = 1 - \tanh^2 \leq 1$). Se $\gamma \|\mathbf{W}_{hh}\| < 1$, il prodotto di $(T - k)$ fattori tende a zero esponenzialmente: il gradiente rispetto a $\mathbf{h}_k$ per $k \ll T$ diventa infinitesimo. La rete non può imparare dipendenze a lungo raggio.

8.3 LSTM — Ogni Gate Spiegato

Motivazione della struttura a gate

Il problema dell'RNN è che un singolo percorso di gradiente (attraverso $\mathbf{W}_{hh}$ ripetuto) causa il vanishing. La soluzione LSTM: creare un percorso **additivo** separato (la cella $\mathbf{c}_t$) e controllare il flusso di informazione con funzioni sigmoid (i gate).

LSTM — EQUAZIONI COMPLETE

Input: $\mathbf{x}_t \in \mathbb{R}^n$, stato precedente $\mathbf{h}_{t-1} \in \mathbb{R}^d$, cella $\mathbf{c}_{t-1} \in \mathbb{R}^d$. Denoteremo $[\mathbf{h}_{t-1}; \mathbf{x}_t] \in \mathbb{R}^{d+n}$ la concatenazione dei due vettori.

Forget gate $\mathbf{f}_t$ — decide quanto dimenticare della cella precedente:

$$\mathbf{f}_t = \sigma(\mathbf{W}_f[\mathbf{h}_{t-1}; \mathbf{x}_t] + \mathbf{b}_f) \in (0, 1)^d$$

Ogni componente di $\mathbf{f}_t$ è vicina a 1 $\rightarrow$ "ricorda", vicina a 0 $\rightarrow$ "dimentica".

Input gate $\mathbf{i}_t$ — decide quanto della nuova informazione memorizzare:

$$\mathbf{i}_t = \sigma(\mathbf{W}_i[\mathbf{h}_{t-1}; \mathbf{x}_t] + \mathbf{b}_i) \in (0, 1)^d$$

Candidato $\tilde{\mathbf{c}}_t$ — nuova informazione candidata da aggiungere:

$$\tilde{\mathbf{c}}_t = \tanh(\mathbf{W}_c[\mathbf{h}_{t-1}; \mathbf{x}_t] + \mathbf{b}_c) \in (-1, 1)^d$$

Aggiornamento della cella — combinazione lineare elemento per elemento:

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t$$

Output gate $\mathbf{o}_t$ — decide cosa trasmettere a $\mathbf{h}_t$:

$$\mathbf{o}_t = \sigma(\mathbf{W}_o[\mathbf{h}_{t-1}; \mathbf{x}_t] + \mathbf{b}_o) \in (0, 1)^d$$

Stato nascosto:

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

Perché LSTM risolve il vanishing gradient

Il gradiente rispetto alla cella al passo precedente è:

$$\frac{\partial \mathbf{c}_t}{\partial \mathbf{c}_{t-1}} = \text{diag}(\mathbf{f}_t)$$

Moltiplicando per tutti i passi da T a k :

$$\frac{\partial \mathbf{c}_T}{\partial \mathbf{c}_k} = \prod_{j=k+1}^T \text{diag}(\mathbf{f}_j)$$

Se il forget gate impara $\mathbf{f}_j \approx \mathbf{1}$ (tenere tutto), il prodotto rimane vicino a $\mathbf{I}$ — il gradiente fluisce senza degradare. La rete sceglie *attivamente* quanto lasciar passare il gradiente. Questa architettura fu chiamata "Constant Error Carousel" da Hochreiter (1997).

8.4 GRU — Versione Semplificata

GRU (Cho et al., 2014) unifica cella e stato nascosto, riducendo i gate da 3 a 2:

GRU — EQUAZIONI

Reset gate — quanto dimenticare dello stato passato per il candidato:

$$\mathbf{r}_t = \sigma(\mathbf{W}_r[\mathbf{h}_{t-1}; \mathbf{x}_t] + \mathbf{b}_r)$$

Update gate — quanto aggiornare lo stato:

$$\mathbf{z}_t = \sigma(\mathbf{W}_z[\mathbf{h}_{t-1}; \mathbf{x}_t] + \mathbf{b}_z)$$

Candidato (usa solo la parte "resettata" di $\mathbf{h}_{t-1}$):

$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}_h[\mathbf{r}_t \odot \mathbf{h}_{t-1}; \mathbf{x}_t] + \mathbf{b}_h)$$

Aggiornamento — interpolazione lineare tra vecchio e nuovo:

$$\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t$$

Quando $\mathbf{z}_t = \mathbf{1}$: $\mathbf{h}_t = \tilde{\mathbf{h}}_t$ (aggiornamento completo). Quando $\mathbf{z}_t = \mathbf{0}$: $\mathbf{h}_t = \mathbf{h}_{t-1}$ (copia identica, nessun aggiornamento). GRU ha il 75% dei parametri dell'LSTM e prestazioni comparabili.

PARTE 9

Transformer e Meccanismi di Attenzione

9.1 Positional Encoding — Derivazione della Proprietà di Traslazione

Perché il Transformer non sa la posizione

Il Transformer calcola l'attenzione tra tutti i token in parallelo — non c'è ricorrenza. Quindi il modello non sa se "gatto" è il primo o il decimo token. Occorre iniettare la posizione esplicitamente aggiungendo un vettore $\mathbf{PE}(t)$ all'embedding del token in posizione t :

$$\mathbf{h}_t^{(0)} = \mathbf{e}_t + \mathbf{PE}(t)$$

Le formule del PE sinusoidale

Per ogni posizione t e ogni dimensione i (con i che va da 0 a $d/2 - 1$):

$$\mathbf{PE}(t, 2i) = \sin\left(\frac{t}{10000^{2i/d}}\right), \quad \mathbf{PE}(t, 2i + 1) = \cos\left(\frac{t}{10000^{2i/d}}\right)$$

Definiamo $\omega_i \triangleq 1/10000^{2i/d}$ — la frequenza angolare per la coppia di dimensioni i . Quindi $\mathbf{PE}(t, 2i) = \sin(\omega_i t)$ e $\mathbf{PE}(t, 2i + 1) = \cos(\omega_i t)$.

Proprietà di traslazione — dimostrazione

Vogliamo mostrare che esiste una matrice $\mathbf{M}_k$ tale che $\mathbf{PE}(t + k) = \mathbf{M}_k \mathbf{PE}(t)$ per ogni t e offset k . Consideriamo la i -esima coppia di dimensioni:

$$\text{1} \quad \mathbf{PE}(t + k, 2i) = \sin(\omega_i(t + k)) = \sin(\omega_i t + \omega_i k)$$

Definizione di PE

$$2 = \sin(\omega_i t) \cos(\omega_i k) + \cos(\omega_i t) \sin(\omega_i k)$$

Formula di addizione del seno: $\sin(\alpha + \beta) = \sin \alpha \cos \beta + \cos \alpha \sin \beta$

$$3 = \cos(\omega_i k) \cdot \mathbf{PE}(t, 2i) + \sin(\omega_i k) \cdot \mathbf{PE}(t, 2i + 1)$$

Sostituzione con la definizione di PE

Analogamente per

$$\mathbf{PE}(t + k, 2i + 1) = -\sin(\omega_i k) \cdot \mathbf{PE}(t, 2i) + \cos(\omega_i k) \cdot \mathbf{PE}(t, 2i + 1).$$

Quindi per la coppia $(2i, 2i + 1)$:

$$\begin{pmatrix} \mathbf{PE}(t + k, 2i) \\ \mathbf{PE}(t + k, 2i + 1) \end{pmatrix} = \begin{pmatrix} \cos(\omega_i k) & \sin(\omega_i k) \\ -\sin(\omega_i k) & \cos(\omega_i k) \end{pmatrix} \begin{pmatrix} \mathbf{PE}(t, 2i) \\ \mathbf{PE}(t, 2i + 1) \end{pmatrix}$$

È una rotazione di angolo $\omega_i k$! La matrice $\mathbf{M}_k$ è la composizione di queste rotazioni per tutte le coppie i — una matrice ortogonale. Questo permette al modello di rappresentare la posizione relativa $(t_1 - t_2)$ attraverso semplici operazioni lineari.

9.2 Self-Attention — Derivazione Completa

Perché il prodotto scalare misura la compatibilità

Se le componenti di $\mathbf{q}$ e $\mathbf{k}$ sono i.i.d. con media 0 e varianza 1, calcoliamo la varianza del prodotto scalare $s = \mathbf{q}^T \mathbf{k} = \sum_{i=1}^{d_k} q_i k_i$:

$$1 \quad \mathbb{E}[q_i k_i] = \mathbb{E}[q_i] \cdot \mathbb{E}[k_i] = 0 \cdot 0 = 0 \text{ (per indipendenza)}$$

Media del prodotto di variabili indipendenti è il prodotto delle medie

$$2 \quad \text{Var}(q_i k_i) = \mathbb{E}[q_i^2 k_i^2] - (\mathbb{E}[q_i k_i])^2 = \mathbb{E}[q_i^2] \cdot \mathbb{E}[k_i^2] - 0 = 1 \cdot 1 = 1$$

$\mathbb{E}[q_i^2] = \text{Var}(q_i) = 1$ per ipotesi; per indipendenza $\mathbb{E}[q_i^2 k_i^2] = \mathbb{E}[q_i^2] \mathbb{E}[k_i^2]$

$$3 \quad \text{Var}(s) = \text{Var}\left(\sum_{i=1}^{d_k} q_i k_i\right) = \sum_{i=1}^{d_k} \text{Var}(q_i k_i) = d_k$$

La varianza di una somma di variabili indipendenti è la somma delle varianze

$$4 \quad \text{std}(s) = \sqrt{d_k}, \text{ quindi dividendo per } \sqrt{d_k}: \text{std}(s/\sqrt{d_k}) = 1$$

Normalizzazione: $\text{Var}(aX) = a^2 \text{Var}(X)$, con $a = 1/\sqrt{d_k}$ si ha varianza $d_k/d_k = 1$

La formula completa

Dati n token con vettori impilati in $\mathbf{X} \in \mathbb{R}^{n \times d}$, le proiezioni sono $\mathbf{Q} = \mathbf{XW}^Q$, $\mathbf{K} = \mathbf{XW}^K$, $\mathbf{V} = \mathbf{XW}^V$:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \underbrace{\text{softmax}\left(\frac{\mathbf{QK}^T}{\sqrt{d_k}}\right)}_{\mathbf{A} \in \mathbb{R}^{n \times n}} \mathbf{V}$$

La matrice $\mathbf{A}_{ij}$ è il peso con cui il token i attende al token j . Ogni riga somma a 1 (per la softmax). Il prodotto $\mathbf{AV}$ produce per ogni token i una media pesata di tutti i value: $\mathbf{z}_i = \sum_j A_{ij} \mathbf{v}_j$.

Masked Self-Attention (per modelli autoregressivi)

Un modello autoregressivo deve predire x_t vedendo solo $x_1, \dots, x_{t-1}$. Si impone che $A_{ij} = 0$ per $j > i$ aggiungendo $-\infty$ prima della softmax:

$$\mathbf{S}_{ij} = \begin{cases} \frac{\mathbf{q}_i^T \mathbf{k}_j}{\sqrt{d_k}} & \text{se } j \leq i \\ -\infty & \text{se } j > i \end{cases}$$

$$\mathbf{A} = \text{softmax}(\mathbf{S}) \quad \Rightarrow \quad A_{ij} = 0 \text{ se } j > i$$

Poiché $e^{-\infty} = 0$, i termini mascherati danno contributo zero nella softmax.

9.3 Multi-Head Attention — Perché Più Teste

Una singola attention può imparare un solo tipo di relazione. Con h teste in parallelo, ogni testa può specializzarsi: una per relazioni grammaticali, una per riferimenti anaforici, una per prossimità, ecc.

MULTI-HEAD ATTENTION

Per $i = 1, \dots, h$, con $d_k = d_v = d/h$:

$$\text{head}_i = \text{Attention}(\mathbf{X}\mathbf{W}_i^Q, \mathbf{X}\mathbf{W}_i^K, \mathbf{X}\mathbf{W}_i^V) \in \mathbb{R}^{n \times d_v}$$

$$\text{MultiHead}(\mathbf{X}) = \text{Concat}(\text{head}_1, \dots, \text{head}_h) \mathbf{W}^O$$

La concatenazione di h vettori in $\mathbb{R}^{d_v}$ produce un vettore in $\mathbb{R}^{h \cdot d_v} = \mathbb{R}^d$.

$\mathbf{W}^O \in \mathbb{R}^{d \times d}$ remappa nello spazio originale. Ogni testa opera in dimensione d/h , quindi il costo totale è identico a una singola attention in dimensione d .

9.4 Il Blocco Transformer Completo

Un layer Transformer (variante Pre-LayerNorm, usata nei modelli moderni) è:

TRANSFORMER LAYER

$$\mathbf{X}' = \mathbf{X} + \text{MultiHead}(\text{LN}(\mathbf{X})) \quad (\text{attention} + \text{residual})$$

$$\mathbf{X}'' = \mathbf{X}' + \text{FFN}(\text{LN}(\mathbf{X}')) \quad (\text{FFN} + \text{residual})$$

con $\text{FFN}(\mathbf{x}) = f(\mathbf{x}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2$ e $d_{\text{ff}} = 4d$.

La Layer Normalization viene applicata *prima* di ogni sublayer (Pre-LN). I residual connection (i "+") assicurano la propagazione del gradiente come discusso in 7.3.

La loss di preaddestramento per modelli autoregressivi (GPT-style) è la cross-entropy sul token successivo:

$$\mathcal{L} = -\frac{1}{N} \sum_{t=1}^N \log P_{\theta}(x_t \mid x_1, \dots, x_{t-1})$$

PARTE 10

Teoria dell'Ottimizzazione

10.1 Funzioni L-Smooth e Convergenza — Dimostrazione Completa

Definizione di L-smooth

Una funzione $f : \mathbb{R}^n \rightarrow \mathbb{R}$ è **L-smooth** se il suo gradiente è Lipschitz con costante L :

$$\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\| \leq L\|\mathbf{x} - \mathbf{y}\| \quad \forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^n$$

Intuitivamente: il gradiente non può cambiare troppo rapidamente. L è il massimo autovalore della matrice Hessiana $\nabla^2 f$ (la matrice delle derivate seconde).

Conseguenza: disuguaglianza quadratica

Da L -smooth segue che f è sovrastimabile da una parabola:

$$f(\mathbf{y}) \leq f(\mathbf{x}) + \nabla f(\mathbf{x})^T (\mathbf{y} - \mathbf{x}) + \frac{L}{2} \|\mathbf{y} - \mathbf{x}\|^2$$

Derivazione breve: Scriviamo $f(\mathbf{y}) - f(\mathbf{x}) = \int_0^1 \nabla f(\mathbf{x} + t(\mathbf{y} - \mathbf{x}))^T (\mathbf{y} - \mathbf{x}) dt$. Sottraendo $\nabla f(\mathbf{x})^T (\mathbf{y} - \mathbf{x})$ e applicando la disug. di Cauchy-Schwarz e L -smoothness si ottiene il bound quadratico.

Teorema di convergenza di GD — ogni passo

Con $\mathbf{x}_{t+1} = \mathbf{x}_t - \frac{1}{L} \nabla f(\mathbf{x}_t)$ (gradient descent con $\alpha = 1/L$), dimostriamo la diminuzione della loss:

- 1 Applicare la disuguaglianza quadratica con $\mathbf{x} = \mathbf{x}_t$ e

$$\mathbf{y} = \mathbf{x}_{t+1} = \mathbf{x}_t - \frac{1}{L} \nabla f(\mathbf{x}_t):$$

$$f(\mathbf{x}_{t+1}) \leq f(\mathbf{x}_t) + \nabla f(\mathbf{x}_t)^T \left(-\frac{1}{L} \nabla f(\mathbf{x}_t)\right) + \frac{L}{2} \left\| \frac{1}{L} \nabla f(\mathbf{x}_t) \right\|^2$$

$$\text{Sostituzione in } f(\mathbf{y}) \leq f(\mathbf{x}) + \nabla f(\mathbf{x})^T (\mathbf{y} - \mathbf{x}) + \frac{L}{2} \|\mathbf{y} - \mathbf{x}\|^2$$

- 2 $= f(\mathbf{x}_t) - \frac{1}{L} \|\nabla f(\mathbf{x}_t)\|^2 + \frac{L}{2} \cdot \frac{1}{L^2} \|\nabla f(\mathbf{x}_t)\|^2 = f(\mathbf{x}_t) - \frac{1}{2L} \|\nabla f(\mathbf{x}_t)\|^2$
 $\nabla f^T \left(-\frac{1}{L} \nabla f\right) = -\frac{1}{L} \|\nabla f\|^2$ e $\frac{L}{2} \cdot \frac{1}{L^2} = \frac{1}{2L}$; combinando: $-\frac{1}{L} + \frac{1}{2L} = -\frac{1}{2L}$

$$f(\mathbf{x}_{t+1}) \leq f(\mathbf{x}_t) - \frac{1}{2L} \|\nabla f(\mathbf{x}_t)\|^2$$

La loss *decreste strettamente* ad ogni passo (a meno che $\nabla f = 0$ — punto critico).
Sommando su $t = 0, \dots, T - 1$ e usando la convessità si ottiene il rate $O(1/T)$.

10.2 Loss Landscape — Hessiana e Geometria

La matrice Hessiana $\mathbf{H} = \nabla_{\theta}^2 \mathcal{L}(\theta)$ ha elementi $H_{ij} = \partial^2 \mathcal{L} / \partial \theta_i \partial \theta_j$. I suoi autovalori $\lambda_1 \leq \dots \leq \lambda_{|\theta|}$ descrivono la curvatura:

Tipo di punto critico ($\nabla \mathcal{L} = 0$)	Condizione sugli autovalori	Significato
Minimo locale	Tutti $\lambda_i > 0$ ($\mathbf{H}$ definita positiva)	La loss cresce in ogni direzione
Massimo locale	Tutti $\lambda_i < 0$ ($\mathbf{H}$ definita negativa)	La loss decresce in ogni direzione
Punto di sella	Sia $\lambda_i > 0$ che $\lambda_j < 0$	Alcune direzioni salgono, altre scendono

In 10^9 dimensioni, la probabilità che *tutti* gli autovalori siano positivi (minimo locale) è estremamente bassa. La maggior parte dei punti con gradiente nullo sono punti di sella — problematici perché il gradiente è zero ma non siamo al minimo.

10.3 Learning Rate Schedules

Lo schedule più usato nei grandi modelli linguistici è il **cosine decay con warmup**:

$$\alpha_t = \begin{cases} \alpha_{\max} \cdot \frac{t}{T_w} & t \leq T_w \text{ (warmup)} \\ \alpha_{\min} + \frac{\alpha_{\max} - \alpha_{\min}}{2} \left(1 + \cos \left(\pi \frac{t - T_w}{T - T_w} \right) \right) & t > T_w \end{cases}$$

T_w è la durata del warmup (tipicamente 1-5% del training totale). Il warmup è necessario perché all'inizio le stime di Adam ($m_0 = v_0 = 0$) non sono affidabili — un learning rate inizialmente alto causerebbe aggiornamenti erratici.

PARTE 11

Statistica Bayesiana Applicata alle Reti Neurali

11.1 Il Teorema di Bayes — Da Zero

La probabilità condizionata di un evento A dato B è definita come:

$$P(A|B) = \frac{P(A \cap B)}{P(B)} \quad (\text{assumendo } P(B) > 0)$$

Analogamente: $P(B|A) = P(A \cap B)/P(A)$. Combinando le due:

$$P(A|B) = \frac{P(B|A) P(A)}{P(B)}$$

Per variabili continue, P diventa densità di probabilità e si scrive p .

Applicazione ai pesi della rete

Nel framework Bayesiano, i pesi θ sono trattati come variabili aleatorie. Abbiamo:

TEOREMA DI BAYES APPLICATO ALL'APPRENDIMENTO

$$\underbrace{p(\theta|\mathcal{D})}_{\text{posterior}} = \frac{\overbrace{p(\mathcal{D}|\theta)}^{\text{likelihood}} \cdot \overbrace{p(\theta)}^{\text{prior}}}{\underbrace{p(\mathcal{D})}_{\text{evidence}}}$$

- **Prior** $p(\theta)$: la nostra credenza sui pesi prima di vedere i dati
- **Likelihood** $p(\mathcal{D}|\theta)$: quanto è probabile osservare i dati se i pesi fossero θ
- **Posterior** $p(\theta|\mathcal{D})$: la credenza aggiornata dopo aver visto i dati
- **Evidence** $p(\mathcal{D}) = \int p(\mathcal{D}|\theta)p(\theta) d\theta$: costante rispetto a θ

Il problema: l'evidence $p(\mathcal{D})$ richiede un integrale su tutti i possibili valori dei pesi — con miliardi di parametri, questo è computazionalmente impossibile.

11.2 KL Divergence — Derivazione da Zero

Vogliamo misurare quanto due distribuzioni q e p siano diverse. La KL divergence è:

$$D_{\text{KL}}(q||p) \triangleq \int q(\theta) \log \frac{q(\theta)}{p(\theta)} d\theta = \mathbb{E}_q[\log q(\theta) - \log p(\theta)]$$

Perché è sempre ≥ 0 ? Usiamo la **disuguaglianza di Jensen**: per una funzione convessa ϕ vale $\phi(\mathbb{E}[X]) \leq \mathbb{E}[\phi(X)]$. Il logaritmo è concavo, quindi $-\log$ è convessa:

$$1 \quad D_{\text{KL}}(q||p) = \mathbb{E}_q \left[\log \frac{q(\theta)}{p(\theta)} \right] = \mathbb{E}_q \left[-\log \frac{p(\theta)}{q(\theta)} \right]$$

Proprietà del logaritmo: $\log(a/b) = -\log(b/a)$

$$2 \quad \geq -\log \mathbb{E}_q \left[\frac{p(\theta)}{q(\theta)} \right]$$

Jensen: $\mathbb{E}[-\log X] \geq -\log \mathbb{E}[X]$ poiché $-\log$ è convessa

$$3 \quad \mathbb{E}_q \left[\frac{p(\theta)}{q(\theta)} \right] = \int q(\theta) \cdot \frac{p(\theta)}{q(\theta)} d\theta = \int p(\theta) d\theta = 1$$

Definizione di valore atteso; l'integrale di una densità è 1

$$4 \quad D_{\text{KL}}(q||p) \geq -\log(1) = 0$$

$$\log(1) = 0$$

L'uguaglianza vale sse $q(\theta) = p(\theta)$ quasi ovunque.

11.3 ELBO — Derivazione Algebrica Completa

Poiché $p(\theta|\mathcal{D})$ è intrattabile, la sostituiamo con $q_\phi(\theta)$ (una famiglia parametrica semplice, es. gaussiane indipendenti) e minimizziamo $D_{\text{KL}}(q_\phi||p(\cdot|\mathcal{D}))$.

$$1 \quad D_{\text{KL}}(q_\phi||p(\cdot|\mathcal{D})) = \mathbb{E}_{q_\phi} [\log q_\phi(\theta) - \log p(\theta|\mathcal{D})]$$

Definizione di KL divergence

$$2 \quad \text{Sostituiamo } p(\theta|\mathcal{D}) = p(\mathcal{D}|\theta)p(\theta)/p(\mathcal{D}):$$

$$= \mathbb{E}_{q_\phi} [\log q_\phi(\theta) - \log p(\mathcal{D}|\theta) - \log p(\theta) + \log p(\mathcal{D})]$$

Teorema di Bayes; $\log(ab/c) = \log a + \log b - \log c$

$$3 \quad = \mathbb{E}_{q_\phi} [\log q_\phi(\theta) - \log p(\theta)] - \mathbb{E}_{q_\phi} [\log p(\mathcal{D}|\theta)] + \log p(\mathcal{D})$$

$\log p(\mathcal{D})$ è costante (non dipende da θ), esce dall'aspettativa

$$4 \quad = D_{\text{KL}}(q_\phi||p) - \mathbb{E}_{q_\phi} [\log p(\mathcal{D}|\theta)] + \log p(\mathcal{D})$$

Il primo termine è la KL tra q_ϕ e il prior $p(\theta)$

5 Riorganizzando:

$$\log p(\mathcal{D}) = \underbrace{\mathbb{E}_{q_\phi}[\log p(\mathcal{D}|\theta)] - D_{\text{KL}}(q_\phi \| p)}_{\mathcal{L}_{\text{ELBO}}(\phi)} + D_{\text{KL}}(q_\phi \| p(\cdot|\mathcal{D}))$$

Spostamento di termini

Poiché $D_{\text{KL}} \geq 0$, abbiamo $\log p(\mathcal{D}) \geq \mathcal{L}_{\text{ELBO}}$. Massimizzare l'ELBO è equivalente a minimizzare $D_{\text{KL}}(q_\phi \| p(\cdot|\mathcal{D}))$:

$$\mathcal{L}_{\text{ELBO}}(\phi) = \underbrace{\mathbb{E}_{q_\phi(\theta)}[\log p(\mathcal{D}|\theta)]}_{\text{termine di ricostruzione}} - \underbrace{D_{\text{KL}}(q_\phi(\theta) \| p(\theta))}_{\text{termine di regolarizzazione}}$$

Il primo termine massimizza la verosimiglianza dei dati sotto q_ϕ — vuole che q_ϕ spieghi i dati. Il secondo minimizza la distanza dal prior — evita che q_ϕ si allontani troppo dalle credenze iniziali.

KL tra due gaussiane — formula chiusa

Per $q = \mathcal{N}(\mu, \sigma^2)$ e $p = \mathcal{N}(0, \sigma_0^2)$, il calcolo esplicito dell'integrale dà:

$$D_{\text{KL}}(\mathcal{N}(\mu, \sigma^2) \| \mathcal{N}(0, \sigma_0^2)) = \frac{1}{2} \left(\frac{\mu^2 + \sigma^2}{\sigma_0^2} - 1 - \log \frac{\sigma^2}{\sigma_0^2} \right)$$

Reparametrization Trick

Per ottimizzare l'ELBO rispetto a $\phi = (\mu, \sigma)$ via backprop, occorre differenziare attraverso il campionamento $\theta \sim q_\phi$. Il campionamento non è differenziabile rispetto ai parametri. Il trick: riscrivere il campionamento come trasformazione differenziabile:

$$\theta = \mu + \sigma \cdot \varepsilon, \quad \varepsilon \sim \mathcal{N}(0, 1)$$

$$\frac{\partial \theta}{\partial \mu} = 1, \quad \frac{\partial \theta}{\partial \sigma} = \varepsilon$$

Il termine stocastico ε non dipende da ϕ , quindi possiamo backpropagare:

$$\nabla_{\phi} \mathbb{E}_{q_{\phi}}[f(\theta)] = \mathbb{E}_{\varepsilon}[\nabla_{\phi} f(\mu + \sigma \varepsilon)].$$

11.4 Reti Neurali Bayesiane

Una **BNN** assegna una distribuzione a ogni peso invece di un singolo valore. Con mean-field VI (gaussiane indipendenti per ogni peso):

$$q_{\phi}(\theta) = \prod_k \mathcal{N}(\mu_k, \sigma_k^2)$$

Ogni peso ha due parametri da imparare: μ_k e σ_k . In pratica si ottimizza

$\rho_k = \log(\exp(\sigma_k) + 1)$ — la "softplus" — per garantire $\sigma_k > 0$. La predizione bayesiana integra su tutti i pesi:

$$p(y^* | \mathbf{x}^*, \mathcal{D}) \approx \frac{1}{T} \sum_{t=1}^T p(y^* | \mathbf{x}^*, \theta^{(t)}), \quad \theta^{(t)} \sim q_{\phi}$$

Questa stima Monte Carlo dell'integrale bayesiano fornisce sia la predizione che la sua incertezza.

11.5 Gaussian Processes e Connessione con le Reti Neurali

Definizione

Un **Gaussian Process** (GP) è una distribuzione su funzioni $f : \mathbb{R}^n \rightarrow \mathbb{R}$. È completamente specificato da una funzione media $\mu(\mathbf{x})$ e una funzione kernel (covarianza) $k(\mathbf{x}, \mathbf{x}')$:

$$f \sim \mathcal{GP}(\mu(\mathbf{x}), k(\mathbf{x}, \mathbf{x}'))$$

La proprietà chiave: per qualsiasi insieme finito di punti $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)}\}$, il vettore $(f(\mathbf{x}^{(1)}), \dots, f(\mathbf{x}^{(N)}))$ segue una **gaussiana multivariata**.

Predizione GP — derivazione dalla gaussiana condizionale

Dati N punti di training con label $\mathbf{y}$ e un punto test $\mathbf{x}^*$, definiamo:

- $\mathbf{K} \in \mathbb{R}^{N \times N}$ con $K_{ij} = k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$ (covarianza tra training points)
- $\mathbf{k}_* \in \mathbb{R}^N$ con $(k_*)_i = k(\mathbf{x}^*, \mathbf{x}^{(i)})$ (covarianza tra test e training)
- $k_{**} = k(\mathbf{x}^*, \mathbf{x}^*)$ (varianza del test point)

La distribuzione congiunta di $\mathbf{y}$ e $f^* = f(\mathbf{x}^*)$ è gaussiana. Usiamo la formula per il condizionamento di una gaussiana multivariata: se

$$\begin{pmatrix} \mathbf{y} \\ f^* \end{pmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{pmatrix} \mathbf{K} + \sigma_n^2 \mathbf{I} & \mathbf{k}_* \\ \mathbf{k}_*^T & k_{**} \end{pmatrix}\right), \text{ allora:}$$

$$\begin{aligned} f^* | \mathbf{y} &\sim \mathcal{N}(\mu^*, (\sigma^*)^2) \\ \mu^* &= \mathbf{k}_*^T (\mathbf{K} + \sigma_n^2 \mathbf{I})^{-1} \mathbf{y} \\ (\sigma^*)^2 &= k_{**} - \mathbf{k}_*^T (\mathbf{K} + \sigma_n^2 \mathbf{I})^{-1} \mathbf{k}_* \end{aligned}$$

σ_n^2 è la varianza del rumore di osservazione. La media μ^* è la predizione; $(\sigma^*)^2$ è l'incertezza — si riduce man mano che ci sono training points vicini a $\mathbf{x}^*$.

Il teorema di Neal — reti infinite = GP

TEOREMA (NEAL, 1994)

Una rete neurale con un singolo hidden layer di N neuroni, con pesi campionati i.i.d. da un prior $p(\mathbf{w})$, converge in distribuzione a un Gaussian Process nel limite $N \rightarrow \infty$. Il kernel del GP è:

$$k(\mathbf{x}, \mathbf{x}') = \mathbb{E}_{w \sim p(w)} [f(w; \mathbf{x}) f(w; \mathbf{x}')]$$

Intuizione: l'output è la somma di N contributi i.i.d. (uno per neurone) — per il teorema del limite centrale, la somma converge a una gaussiana per $N \rightarrow \infty$.

Inizializzazione dei Pesi — Derivazioni Complete

Perché l'inizializzazione conta, e come sceglierla in modo fondato matematicamente.

12.1 Perché l'Inizializzazione è Critica

Prima del training, i pesi devono essere inizializzati a valori che permettano ai segnali di propagarsi attraverso la rete senza esplodere né annullarsi. Consideriamo cosa succede al forward pass con L layer, ciascuno con n neuroni e pesi i.i.d.

$w \sim \mathcal{N}(0, \sigma_w^2)$ e input $x \sim \mathcal{N}(0, \sigma_x^2)$.

L'output del layer ℓ (ignorando il bias e l'attivazione per semplicità) è

$z_i^{[\ell]} = \sum_{j=1}^n w_{ij}^{[\ell]} a_j^{[\ell-1]}$. La varianza si propaga come:

$$\text{Var}(z_i^{[\ell]}) = n \cdot \sigma_w^2 \cdot \text{Var}(a_j^{[\ell-1]})$$

Dopo L layer: $\text{Var}(z^{[L]}) = (n\sigma_w^2)^L \cdot \text{Var}(x)$. Se $n\sigma_w^2 \neq 1$ questo fattore esplode o implode esponenzialmente in L . L'obiettivo dell'inizializzazione è mantenere $n\sigma_w^2 \approx 1$.

12.2 Xavier / Glorot Initialization

Glorot e Bengio (2010) proposero di bilanciare la varianza sia nel forward pass che nel backward pass.

Condizione per il forward pass

Vogliamo $\text{Var}(\mathbf{z}^{[\ell]}) = \text{Var}(\mathbf{a}^{[\ell-1]})$. Questo richiede (con attivazione lineare):

$$n^{[\ell-1]} \cdot \sigma_w^2 = 1 \implies \sigma_w^2 = \frac{1}{n^{[\ell-1]}}$$

Condizione per il backward pass

Nel backward pass il gradiente si propaga come $\partial \mathcal{L} / \partial \mathbf{a}^{[\ell-1]} = (\mathbf{W}^{[\ell]})^T \boldsymbol{\delta}^{[\ell]}$. Con lo stesso ragionamento:

$$n^{[\ell]} \cdot \sigma_w^2 = 1 \implies \sigma_w^2 = \frac{1}{n^{[\ell]}}$$

Compromesso di Glorot

Le due condizioni si contraddicono quando $n^{[\ell]} \neq n^{[\ell-1]}$. Il compromesso è la media armonica:

$$\sigma_w^2 = \frac{2}{n^{[\ell-1]} + n^{[\ell]}}$$

In pratica si usa la distribuzione uniforme $\mathcal{U}[-a, a]$ con varianza $a^2/3$, da cui:

$$a = \sqrt{\frac{6}{n^{[\ell-1]} + n^{[\ell]}}}$$

ESEMPIO — LAYER CON 128 INPUT E 64 OUTPUT

$$\sigma_w^2 = \frac{2}{128 + 64} = \frac{2}{192} \approx 0.01042, \quad \sigma_w \approx 0.1021$$

$$a = \sqrt{\frac{6}{192}} \approx 0.1768$$

I pesi vengono campionati da $\mathcal{U}[-0.1768, 0.1768]$ oppure $\mathcal{N}(0, 0.1021^2)$.

12.3 He Initialization (per ReLU)

La ReLU azzerà metà dei neuroni (quelli con $z < 0$), dimezzando effettivamente la varianza del segnale. Per compensare, He et al. (2015) raddoppiano σ_w^2 rispetto a Xavier:

Derivazione

Con ReLU, $a = \max(0, z)$. Se $z \sim \mathcal{N}(0, \sigma_z^2)$:

$$1 \quad \mathbb{E}[a] = \mathbb{E}[\max(0, z)] = \int_0^\infty z \cdot \frac{1}{\sigma_z \sqrt{2\pi}} e^{-z^2/2\sigma_z^2} dz = \frac{\sigma_z}{\sqrt{2\pi}}$$

Integrale della semi-gaussiana

$$2 \quad \mathbb{E}[a^2] = \mathbb{E}[\max(0, z)^2] = \int_0^\infty z^2 \cdot \frac{1}{\sigma_z \sqrt{2\pi}} e^{-z^2/2\sigma_z^2} dz = \frac{\sigma_z^2}{2}$$

Metà della varianza di z (la metà positiva)

$$3 \quad \text{Var}(a) = \mathbb{E}[a^2] - (\mathbb{E}[a])^2 = \frac{\sigma_z^2}{2} - \frac{\sigma_z^2}{2\pi} = \sigma_z^2 \left(\frac{1}{2} - \frac{1}{2\pi} \right) \approx \frac{\sigma_z^2}{2}$$

Per grandi σ_z , il termine $1/(2\pi)$ è trascurabile

Quindi ReLU dimezza la varianza: $\text{Var}(a) \approx \text{Var}(z)/2$. Per mantenere $\text{Var}(z^{[\ell]}) = \text{Var}(a^{[\ell-1]})$, occorre compensare:

$$n^{[\ell-1]} \cdot \sigma_w^2 \cdot \frac{1}{2} = 1 \implies \sigma_w^2 = \frac{2}{n^{[\ell-1]}}$$

La distribuzione risultante è $\mathcal{N}(0, 2/n_{\text{in}})$, chiamata **He initialization** o **Kaiming initialization**.

12.4 Orthogonal Initialization

Per RNN, un'inizializzazione specialmente efficace: campionare una matrice da $\mathcal{N}(0, 1)$, fare la decomposizione SVD $\mathbf{M} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$, e usare $\mathbf{U}$ come matrice dei pesi.

Le matrici ortogonali soddisfano $\mathbf{W}^T\mathbf{W} = \mathbf{I}$, quindi tutti gli autovalori hanno modulo 1. Questo garantisce $\|\mathbf{W}\mathbf{h}\| = \|\mathbf{h}\|$ — le norme non esplodono né implodono — condizione ideale per le RNN dove la matrice ricorrente viene applicata T volte.

PARTE 13

Normalizzazioni: Zoo Completo e Confronto

13.1 Le Dimensioni da Normalizzare

Dati i tensori di attivazione $\mathbf{X} \in \mathbb{R}^{N \times C \times H \times W}$ (batch $\times$ canali $\times$ altezza $\times$ larghezza), le diverse normalizzazioni differiscono per le dimensioni su cui si calcolano media e varianza:

Metodo	Dimensioni normalizzate	Usato in
Batch Norm (BN)	N, H, W (per canale C)	CNN, MLP

Layer Norm (LN)	C, H, W (per esempio N)	Transformer, NLP
Instance Norm (IN)	H, W (per esempio N e canale C)	Style transfer
Group Norm (GN)	Gruppi di canali, H, W	Detection, piccoli batch
RMS Norm	Come LN ma senza sottrarre la media	LLaMA, PaLM, Gemma

13.2 Layer Normalization — Derivazione della Backward Pass

Dati d valori $x_1, \dots, x_d$ (le feature di un singolo token), la LN calcola:

$$\mu = \frac{1}{d} \sum_{i=1}^d x_i, \quad \sigma^2 = \frac{1}{d} \sum_{i=1}^d (x_i - \mu)^2, \quad \hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \varepsilon}}, \quad y_i = \gamma_i \hat{x}_i + \beta$$

Gradiente rispetto a x_j — derivazione passo per passo

Questo è il calcolo più complesso della LN. Definiamo $s = \sqrt{\sigma^2 + \varepsilon}$ per brevità.

1 Per la regola della catena: $\frac{\partial \mathcal{L}}{\partial x_j} = \sum_{i=1}^d \frac{\partial \mathcal{L}}{\partial \hat{x}_i} \frac{\partial \hat{x}_i}{\partial x_j}$

Ogni $\hat{x}_i$ dipende da x_j attraverso μ e σ^2

2 $\hat{x}_i = (x_i - \mu)/s$. Deriviamo rispetto a x_j :

$$\frac{\partial \hat{x}_i}{\partial x_j} = \frac{1}{s} \left(\delta_{ij} - \frac{\partial \mu}{\partial x_j} \right) - \frac{x_i - \mu}{s^2} \frac{\partial s}{\partial x_j}$$

Regola del quoziente su $(x_i - \mu)/s$

3 $\frac{\partial \mu}{\partial x_j} = \frac{1}{d}$ (la media dipende ugualmente da tutti gli x_j)

Derivata di $\frac{1}{d} \sum_i x_i$ rispetto a x_j

4 $\frac{\partial \sigma^2}{\partial x_j} = \frac{2}{d}(x_j - \mu)$, quindi $\frac{\partial s}{\partial x_j} = \frac{x_j - \mu}{ds}$

Regola della catena su $\sigma^2 = \frac{1}{d} \sum_i (x_i - \mu)^2$, poi su $s = \sqrt{\sigma^2 + \varepsilon}$

5 Sostituendo e raccogliendo, con $\delta_i = \partial \mathcal{L} / \partial \hat{x}_i = \partial \mathcal{L} / \partial y_i \cdot \gamma_i$:

$$\frac{\partial \mathcal{L}}{\partial x_j} = \frac{1}{s} \left[\delta_j - \frac{1}{d} \sum_{i=1}^d \delta_i - \hat{x}_j \cdot \frac{1}{d} \sum_{i=1}^d \delta_i \hat{x}_i \right]$$

Raccogliendo tutti i termini e riconoscendo le somme

$$\frac{\partial \mathcal{L}}{\partial x_j} = \frac{1}{s} \left[\frac{\partial \mathcal{L}}{\partial \hat{x}_j} - \frac{1}{d} \sum_{i=1}^d \frac{\partial \mathcal{L}}{\partial \hat{x}_i} - \hat{x}_j \cdot \frac{1}{d} \sum_{i=1}^d \frac{\partial \mathcal{L}}{\partial \hat{x}_i} \hat{x}_i \right]$$

13.3 RMSNorm

Zhang e Sennrich (2019) osservarono che sottrarre la media in LN ha effetto trascurabile nelle reti profonde. Propongono di normalizzare solo per la radice della media dei quadrati (RMS):

$$\text{RMS}(\mathbf{x}) = \sqrt{\frac{1}{d} \sum_{i=1}^d x_i^2 + \varepsilon}$$

$$\text{RMSNorm}(\mathbf{x})_i = \frac{x_i}{\text{RMS}(\mathbf{x})} \cdot \gamma_i$$

Vantaggi: (1) nessun calcolo della media — più veloce, (2) invariante alla riscalatura degli input ma non alla traslazione, (3) equivalente a LN quando la media è zero (che tende ad essere vera per layer profondi).

Gradiente di RMSNorm

Con $r = \text{RMS}(\mathbf{x})$, definiamo $\hat{x}_i = x_i/r$. La derivata rispetto a x_j :

$$\frac{\partial r}{\partial x_j} = \frac{x_j}{dr}, \quad \frac{\partial \hat{x}_i}{\partial x_j} = \frac{\delta_{ij}}{r} - \frac{x_i x_j}{r^3 d}$$

$$\frac{\partial \mathcal{L}}{\partial x_j} = \frac{\gamma_j}{r} \frac{\partial \mathcal{L}}{\partial y_j} - \frac{x_j}{r^2 d} \sum_{i=1}^d \frac{\partial \mathcal{L}}{\partial y_i} \gamma_i \hat{x}_i$$

13.4 Group Normalization

Wu e He (2018) dividono i C canali in G gruppi di C/G canali ciascuno, e normalizzano entro ogni gruppo indipendentemente. Per il canale c nel gruppo $g = \lfloor cG/C \rfloor$:

$$\mu_g = \frac{1}{|S_g|} \sum_{(n,c,h,w) \in S_g} x_{nchw}, \quad \sigma_g^2 = \frac{1}{|S_g|} \sum_{(n,c,h,w) \in S_g} (x_{nchw} - \mu_g)^2$$
$$\hat{x}_{nchw} = \frac{x_{nchw} - \mu_g}{\sqrt{\sigma_g^2 + \varepsilon}}, \quad y_{nchw} = \gamma_c \hat{x}_{nchw} + \beta_c$$

Dove $S_g = \{(n, c', h, w) \mid \lfloor c'G/C \rfloor = g\}$. Group Norm è indipendente dal batch size, e diventa Layer Norm per $G = 1$ e Instance Norm per $G = C$.

14.1 Entropia di Shannon

L'**entropia** di una variabile aleatoria discreta X con distribuzione p misura l'incertezza media (o la quantità di informazione media per osservazione):

$$H(X) = - \sum_x p(x) \log_2 p(x) = \mathbb{E}_p[-\log_2 p(X)] \quad [\text{bit}]$$

Con base e invece di 2 si misura in nats. Proprietà fondamentali:

- $H(X) \geq 0$ sempre (poiché $0 \leq p(x) \leq 1 \Rightarrow -\log p(x) \geq 0$)
- $H(X) = 0$ sse X è deterministica (un solo esito ha probabilità 1)
- $H(X) \leq \log_2 |X|$ con uguaglianza sse X è uniforme su $|X|$ valori

ESEMPIO — LANCIO DI MONETA

Moneta equa ($p = 0.5$):

$$H = -(0.5 \log_2 0.5 + 0.5 \log_2 0.5) = -(-0.5 - 0.5) = 1 \text{ bit.}$$

Moneta truccata ($p = 0.9$):

$$H = -(0.9 \log_2 0.9 + 0.1 \log_2 0.1) \approx -(-.137 - .332) = 0.469 \text{ bit.}$$

Meno incertezza perché l'esito è quasi prevedibile.

14.2 Cross-Entropy e la sua Connessione con la Loss

La **cross-entropy** tra una distribuzione reale p e una distribuzione modello q è:

$$H(p, q) = - \sum_x p(x) \log q(x) = \mathbb{E}_p[-\log q(X)]$$

Relazione fondamentale con la KL divergence:

$$H(p, q) = H(p) + D_{\text{KL}}(p||q)$$

Dimostrazione:

- 1 $H(p, q) = - \sum_x p(x) \log q(x) = - \sum_x p(x) \log p(x) + \sum_x p(x) \log \frac{p(x)}{q(x)}$
Aggiungiamo e sottraiamo $\sum_x p(x) \log p(x)$
- 2 $= H(p) + D_{\text{KL}}(p||q)$
Per definizione di entropia e KL divergence

Poiché $H(p)$ è fissa (dipende solo dai dati veri, non dal modello), minimizzare $H(p, q)$ è equivalente a minimizzare $D_{\text{KL}}(p||q)$ — ovvero a fare la MLE. La cross-entropy loss delle reti neurali è esattamente questo.

14.3 Informazione Mutua

L'**informazione mutua** tra X e Y misura quanto osservare una variabile riduce l'incertezza sull'altra:

$$I(X; Y) = \sum_{x,y} p(x, y) \log \frac{p(x, y)}{p(x)p(y)} = H(X) - H(X|Y) = H(Y) - H(Y|X)$$

$I(X; Y) = D_{\text{KL}}(p(x, y) \| p(x)p(y)) \geq 0$, con uguaglianza sse X e Y sono indipendenti.

Information Bottleneck e Deep Learning

Tishby e Schwartz-Ziv (2017) proposero di interpretare le reti neurali tramite il principio dell'Information Bottleneck: ogni layer deve comprimere l'input X ritenendo solo l'informazione rilevante per il target Y . Il layer ottimale massimizza:

$$I(T; Y) - \beta I(T; X)$$

dove T è la rappresentazione del layer, β controlla il trade-off compressione/predizione. Sebbene controversa, questa prospettiva ha generato nuove intuizioni sulla generalizzazione nelle reti profonde.

14.4 MLE come Minimizzazione della Cross-Entropy

Vogliamo mostrare che massimizzare la log-likelihood è equivalente a minimizzare la cross-entropy tra la distribuzione empirica $\hat{p}$ e il modello p_θ .

La distribuzione empirica su N esempi è $\hat{p}(x) = \frac{1}{N} \sum_{i=1}^N \mathbf{1}[x = x^{(i)}]$. Il log-likelihood:

$$\textcircled{1} \quad \hat{\theta} = \arg \max_{\theta} \frac{1}{N} \sum_{i=1}^N \log p_{\theta}(x^{(i)}) = \arg \max_{\theta} \mathbb{E}_{\hat{p}}[\log p_{\theta}(X)]$$

Definizione di MLE

$$\textcircled{2} \quad = \arg \min_{\theta} -\mathbb{E}_{\hat{p}}[\log p_{\theta}(X)] = \arg \min_{\theta} H(\hat{p}, p_{\theta})$$

Definizione di cross-entropy

$$\textcircled{3} \quad = \arg \min_{\theta} D_{\text{KL}}(\hat{p} \| p_{\theta}) \text{ (poiché } H(\hat{p}) \text{ non dipende da } \theta)$$

$$H(\hat{p}, p_\theta) = H(\hat{p}) + D_{\text{KL}}(\hat{p} \| p_\theta)$$

La MLE è equivalente a trovare il modello più vicino (in KL) alla distribuzione empirica dei dati.

PARTE 15

Modelli Generativi: VAE, GAN, Diffusion

15.1 Variational Autoencoder (VAE)

Il modello generativo

Un VAE (Kingma e Welling, 2013) è un modello latente: immagina che ogni dato $\mathbf{x}$ sia stato generato da una variabile latente $\mathbf{z} \in \mathbb{R}^k$ tramite un processo in due passi:

$$\begin{aligned} \mathbf{z} &\sim p(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I}) && \text{(prior sul latente)} \\ \mathbf{x}|\mathbf{z} &\sim p_\theta(\mathbf{x}|\mathbf{z}) && \text{(decoder: rete neurale con parametri } \theta) \end{aligned}$$

Il problema dell'inferenza

Per addestrare il decoder, occorre la posterior $p_\theta(\mathbf{z}|\mathbf{x}) = p_\theta(\mathbf{x}|\mathbf{z})p(\mathbf{z})/p_\theta(\mathbf{x})$. Ma $p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x}|\mathbf{z})p(\mathbf{z}) d\mathbf{z}$ è intrattabile. Si introduce un **encoder** (inference network) $q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\boldsymbol{\mu}_\phi(\mathbf{x}), \text{diag}(\boldsymbol{\sigma}_\phi^2(\mathbf{x})))$ per approssimare la posterior.

ELBO del VAE

$$\mathcal{L}_{\text{VAE}}(\theta, \phi; \mathbf{x}) = \underbrace{\mathbb{E}_{q_{\phi}(\mathbf{z}|\mathbf{x})}[\log p_{\theta}(\mathbf{x}|\mathbf{z})]}_{\text{reconstruction loss}} - \underbrace{D_{\text{KL}}(q_{\phi}(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))}_{\text{regularization}}$$

Con gaussiane, il termine KL ha forma chiusa. Per $q_{\phi}(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\boldsymbol{\mu}, \text{diag}(\boldsymbol{\sigma}^2))$ e $p(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$:

$$D_{\text{KL}} = \frac{1}{2} \sum_{j=1}^k (\mu_j^2 + \sigma_j^2 - 1 - \log \sigma_j^2)$$

Reparametrization trick nel VAE

Il campionamento $\mathbf{z} \sim q_{\phi}(\mathbf{z}|\mathbf{x})$ viene riscritto come:

$$\mathbf{z} = \boldsymbol{\mu}_{\phi}(\mathbf{x}) + \boldsymbol{\sigma}_{\phi}(\mathbf{x}) \odot \boldsymbol{\varepsilon}, \quad \boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

Questo rende il campionamento differenziabile rispetto ai parametri ϕ . Il gradiente dell'ELBO rispetto a ϕ :

$$\nabla_{\phi} \mathcal{L}_{\text{VAE}} = \mathbb{E}_{\boldsymbol{\varepsilon}}[\nabla_{\phi} \log p_{\theta}(\mathbf{x}|\boldsymbol{\mu}_{\phi} + \boldsymbol{\sigma}_{\phi} \odot \boldsymbol{\varepsilon})] - \nabla_{\phi} D_{\text{KL}}$$

15.2 Generative Adversarial Network (GAN)

Il gioco minimax

Un GAN (Goodfellow et al., 2014) ha due reti che competono:

- **Generatore** G_θ : mappa rumore $\mathbf{z} \sim p(\mathbf{z})$ in dati $G_\theta(\mathbf{z}) \sim p_g$
- **Discriminatore** D_ϕ : stima la probabilità che un input sia reale (da p_{data}) vs. generato

$$\min_{\theta} \max_{\phi} V(G_\theta, D_\phi) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} [\log D_\phi(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})} [\log(1 - D_\phi(G_\theta(\mathbf{z})))]$$

Discriminatore ottimale — derivazione

Per un generatore fisso G , il discriminatore ottimale si ottiene massimizzando V puntualmente rispetto a $D(\mathbf{x})$ per ogni $\mathbf{x}$:

- 1 L'obiettivo del discriminatore si può scrivere come integrale:

$$V = \int_{\mathbf{x}} [p_{\text{data}}(\mathbf{x}) \log D(\mathbf{x}) + p_g(\mathbf{x}) \log(1 - D(\mathbf{x}))] d\mathbf{x}$$

Riscrittura delle aspettative come integrali

- 2 Per ogni $\mathbf{x}$, massimizziamo $f(d) = a \log d + b \log(1 - d)$ con

$$a = p_{\text{data}}(\mathbf{x}) \text{ e } b = p_g(\mathbf{x})$$

Ottimizzazione puntuale

- 3 $f'(d) = a/d - b/(1 - d) = 0 \implies a(1 - d) = bd \implies d = a/(a + b)$

Poniamo la derivata uguale a zero e risolviamo

$$D^*(\mathbf{x}) = \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_g(\mathbf{x})}$$

Loss globale all'equilibrio — connessione con la JSD

Sostituendo D^* nell'obiettivo del generatore:

$$1 \quad C(G) = \mathbb{E}_{p_{\text{data}}} \left[\log \frac{p_{\text{data}}}{p_{\text{data}} + p_g} \right] + \mathbb{E}_{p_g} \left[\log \frac{p_g}{p_{\text{data}} + p_g} \right]$$

*Sostituzione di D^**

$$2 \quad = -\log 4 + D_{\text{KL}} \left(p_{\text{data}} \parallel \frac{p_{\text{data}} + p_g}{2} \right) + D_{\text{KL}} \left(p_g \parallel \frac{p_{\text{data}} + p_g}{2} \right)$$

Riscrittura della KL aggiungendo e sottraendo $\log 2$

$$3 \quad = -\log 4 + 2 \text{JSD}(p_{\text{data}} \parallel p_g)$$

Definizione di Jensen-Shannon Divergence:

$$\text{JSD}(p \parallel q) = \frac{1}{2} D_{\text{KL}}(p \parallel m) + \frac{1}{2} D_{\text{KL}}(q \parallel m) \text{ con } m = (p + q)/2$$

Il minimo globale $C(G) = -\log 4$ si raggiunge sse $p_g = p_{\text{data}}$, cioè il generatore produce la stessa distribuzione dei dati reali.

15.3 Diffusion Models

Il processo forward (aggiunta di rumore)

I modelli di diffusione (Ho et al., 2020) aggiungono rumore gaussiano a un'immagine $\mathbf{x}_0$ in T passi, con varianze $\beta_1 < \beta_2 < \dots < \beta_T$:

$$q(\mathbf{x}_t | \mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1 - \beta_t} \mathbf{x}_{t-1}, \beta_t \mathbf{I})$$

Definiamo $\alpha_t = 1 - \beta_t$ e $\bar{\alpha}_t = \prod_{s=1}^t \alpha_s$. La distribuzione marginale ad ogni passo ha forma chiusa:

$$q(\mathbf{x}_t | \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t} \mathbf{x}_0, (1 - \bar{\alpha}_t) \mathbf{I})$$

Ovvero: $\mathbf{x}_t = \sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \boldsymbol{\varepsilon}$ con $\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. Per T grande, $\bar{\alpha}_T \approx 0$ e $\mathbf{x}_T \approx \mathcal{N}(\mathbf{0}, \mathbf{I})$.

Il processo reverse (denoising)

Una rete neurale $\boldsymbol{\varepsilon}_\theta(\mathbf{x}_t, t)$ impara a predire il rumore aggiunto in ogni passo. La loss di training è:

$$\mathcal{L}_{\text{simple}} = \mathbb{E}_{t, \mathbf{x}_0, \boldsymbol{\varepsilon}} \left[\|\boldsymbol{\varepsilon} - \boldsymbol{\varepsilon}_\theta(\sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \boldsymbol{\varepsilon}, t)\|^2 \right]$$

Questo è derivato dall'ELBO del modello di diffusione, ma la semplificazione di Ho et al. mostra che pesare uniformemente tutti i passi temporali funziona meglio in pratica.

Campionamento (DDPM)

Partendo da $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, si applica il reverse process per $t = T, T - 1, \dots, 1$:

$$\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \boldsymbol{\varepsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}, \quad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

con $\sigma_t^2 = \beta_t$ o $\sigma_t^2 = \frac{(1 - \bar{\alpha}_{t-1})\beta_t}{1 - \bar{\alpha}_t}$ (le due scelte sono equivalenti per modelli perfetti).

Apprendimento per Rinforzo: RL e Policy Gradient

16.1 Markov Decision Process (MDP)

Un **MDP** è una tupla $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$ dove:

- $\mathcal{S}$: spazio degli stati
- $\mathcal{A}$: spazio delle azioni
- $P(s'|s, a)$: probabilità di transizione al passo $t + 1$ nello stato s' da s prendendo azione a
- $R(s, a) \in \mathbb{R}$: ricompensa immediata
- $\gamma \in [0, 1)$: fattore di sconto

L'agente segue una **policy** $\pi_\theta(a|s)$ (distribuzione sulle azioni dato lo stato, parametrizzata da θ). L'obiettivo è massimizzare il **return atteso**:

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^T \gamma^t R(s_t, a_t) \right] = \mathbb{E}_{\tau \sim \pi_\theta} [G(\tau)]$$

dove $\tau = (s_0, a_0, r_0, s_1, a_1, \dots)$ è una traiettoria.

16.2 Equazioni di Bellman

La **funzione valore** $V^\pi(s)$ è il return atteso partendo da s e seguendo π :

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid s_0 = s \right]$$

L'**equazione di Bellman** esprime V^π ricorsivamente:

$$V^\pi(s) = \sum_a \pi(a|s) \sum_{s'} P(s'|s, a) [R(s, a) + \gamma V^\pi(s')]$$

La **funzione Q** $Q^\pi(s, a)$ è il return atteso partendo da s , prendendo azione a , poi seguendo π :

$$Q^\pi(s, a) = R(s, a) + \gamma \sum_{s'} P(s'|s, a) V^\pi(s'), \quad V^\pi(s) = \sum_a \pi(a|s) Q^\pi(s, a)$$

16.3 Policy Gradient Theorem — Dimostrazione

Come calcolare $\nabla_\theta J(\theta)$? La difficoltà è che sia la policy che la distribuzione delle traiettorie dipendono da θ .

1 $J(\theta) = \int_\tau p_\theta(\tau) G(\tau) d\tau$ dove

$$p_\theta(\tau) = p(s_0) \prod_{t=0}^T \pi_\theta(a_t|s_t) P(s_{t+1}|s_t, a_t)$$

Scrittura esplicita della speranza come integrale

2 $\nabla_\theta J = \int_\tau \nabla_\theta p_\theta(\tau) G(\tau) d\tau$

Scambio derivata-integrale (sotto condizioni di regolarità)

3 Usiamo il "log-derivative trick": $\nabla_\theta p_\theta(\tau) = p_\theta(\tau) \nabla_\theta \log p_\theta(\tau)$

Identità $\nabla \log f = \nabla f / f \Rightarrow \nabla f = f \nabla \log f$

$$4 \quad \nabla_{\theta} \log p_{\theta}(\tau) = \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$$

Poiché $\log p(s_0)$ e $\log P(s_{t+1} | s_t, a_t)$ non dipendono da θ

$$5 \quad \nabla_{\theta} J = \mathbb{E}_{\tau \sim \pi_{\theta}} \left[G(\tau) \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right]$$

Raccogliendo nell'aspettativa

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\sum_{t=0}^T G(\tau) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right]$$

Questo è il **Policy Gradient Theorem**. Non richiede di differenziare attraverso l'ambiente — solo attraverso la policy π_{θ} .

16.4 REINFORCE

L'algoritmo REINFORCE (Williams, 1992) stima il gradiente con Monte Carlo su traiettorie campionate:

$$\nabla_{\theta} J \approx \frac{1}{N} \sum_{n=1}^N \sum_{t=0}^T G_t^{(n)} \nabla_{\theta} \log \pi_{\theta}(a_t^{(n)} | s_t^{(n)})$$

dove $G_t = \sum_{t'=t}^T \gamma^{t'-t} R(s_{t'}, a_{t'})$ è il return dal passo t in poi (causality trick — le azioni passate non influenzano le ricompense future).

Baseline per ridurre la varianza

L'estimatore REINFORCE ha alta varianza. Sottraendo una baseline $b(s_t)$ (che non introduce bias se è funzione solo dello stato):

$$\nabla_{\theta} J \approx \mathbb{E} \left[\sum_{t=0}^T (G_t - b(s_t)) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right]$$

La scelta ottimale di baseline è la funzione valore $b(s_t) = V^{\pi}(s_t)$. La differenza $A_t = G_t - V^{\pi}(s_t)$ è l'**advantage function**: misura quanto l'azione presa sia migliore della media.

16.5 PPO (Proximal Policy Optimization)

PPO (Schulman et al., 2017) limita l'aggiornamento della policy per evitare passi troppo grandi che destabilizzano il training. Definendo il rapporto di importanza $r_t(\theta) = \pi_{\theta}(a_t | s_t) / \pi_{\theta_{\text{old}}}(a_t | s_t)$:

$$\mathcal{L}^{\text{CLIP}}(\theta) = \mathbb{E}_t [\min(r_t(\theta) A_t, \text{clip}(r_t(\theta), 1 - \varepsilon, 1 + \varepsilon) A_t)]$$

Il clip impedisce che r_t si allontani troppo da 1 (la policy non cambia troppo in un aggiornamento). Con $\varepsilon = 0.2$, l'aggiornamento è limitato al $\pm 20\%$ del peso della policy precedente.

PARTE 17

Graph Neural Networks (GNN)

17.1 Grafi come Struttura Dati

Un grafo $G = (V, E)$ ha $|V| = N$ nodi e $|E|$ archi. Si rappresenta con la **matrice di adiacenza** $\mathbf{A} \in \{0, 1\}^{N \times N}$ (con $A_{ij} = 1$ se esiste l'arco (i, j)) e la matrice delle feature $\mathbf{X} \in \mathbb{R}^{N \times d}$ (feature di ogni nodo).

17.2 Message Passing Framework

L'idea centrale: ogni nodo aggrega le informazioni dai suoi vicini, aggiorna la propria rappresentazione, e ripete per K layer:

MESSAGE PASSING — PASSO GENERICO AL LAYER K

Aggregazione — il nodo v raccoglie messaggi dai vicini $\mathcal{N}(v)$:

$$\mathbf{m}_v^{(k)} = \text{AGG}\left(\{h_u^{(k-1)} : u \in \mathcal{N}(v)\}\right)$$

Aggiornamento — combina il messaggio aggregato con la rappresentazione corrente:

$$h_v^{(k)} = \text{UPDATE}\left(h_v^{(k-1)}, \mathbf{m}_v^{(k)}\right)$$

Inizializzazione: $h_v^{(0)} = \mathbf{x}_v$ (feature del nodo). L'output finale $h_v^{(K)}$ è la rappresentazione appresa del nodo v nel contesto del suo vicinato a K hop.

17.3 Graph Convolutional Network (GCN)

GCN (Kipf e Welling, 2017) usa una convoluzione spettrale semplificata. La propagazione al layer k è:

$$\mathbf{H}^{(k)} = \sigma\left(\hat{\mathbf{A}} \mathbf{H}^{(k-1)} \mathbf{W}^{(k)}\right)$$

dove $\hat{\mathbf{A}} = \tilde{\mathbf{D}}^{-1/2} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-1/2}$ è la matrice di adiacenza normalizzata, con $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ (auto-loop) e $\tilde{\mathbf{D}}_{ii} = \sum_j \tilde{A}_{ij}$ (matrice dei gradi).

Perché la normalizzazione simmetrica

Senza normalizzazione, nodi ad alto grado riceverebbero messaggi amplificati, causando instabilità numerica. La normalizzazione $\tilde{\mathbf{D}}^{-1/2} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-1/2}$ garantisce che tutti gli autovalori di $\hat{\mathbf{A}}$ siano in $[-1, 1]$, rendendo il sistema stabile.

Esplicitamente, l'elemento (i, j) di $\hat{\mathbf{A}}$ è:

$$\hat{A}_{ij} = \frac{\tilde{A}_{ij}}{\sqrt{\tilde{d}_i} \sqrt{\tilde{d}_j}}$$

Quindi l'aggiornamento del nodo i è: $h_i^{(k)} = \sigma\left(\sum_{j \in \tilde{\mathcal{N}}(i)} \frac{1}{\sqrt{\tilde{d}_i} \sqrt{\tilde{d}_j}} h_j^{(k-1)} \mathbf{W}^{(k)}\right)$

17.4 Graph Attention Network (GAT)

GAT (Veličković et al., 2018) impara pesi di attenzione diversi per ogni arco, permettendo al nodo di "concentrarsi" sui vicini più rilevanti:

$$e_{ij} = a(\mathbf{W}\mathbf{h}_i, \mathbf{W}\mathbf{h}_j) = \text{LeakyReLU}(\mathbf{a}^T [\mathbf{W}\mathbf{h}_i \parallel \mathbf{W}\mathbf{h}_j])$$

$$\alpha_{ij} = \text{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}(i)} \exp(e_{ik})}$$

$$h'_i = \sigma \left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij} \mathbf{W} \mathbf{h}_j \right)$$

Il vettore $\mathbf{a} \in \mathbb{R}^{2d'}$ è un parametro appreso che determina il meccanismo di attenzione. Con multi-head attention (K teste): $h'_i = \parallel_{k=1}^K \sigma \left(\sum_j \alpha_{ij}^k \mathbf{W}^k \mathbf{h}_j \right)$.

PARTE 18

Metric Learning e Apprendimento Contrastivo

18.1 Contrastive Loss

L'obiettivo del metric learning è imparare uno spazio di embedding in cui esempi simili sono vicini e dissimili sono lontani. Data una rete $f_\theta : \mathbb{R}^n \rightarrow \mathbb{R}^d$ e coppie (x_i, x_j) con etichetta $y_{ij} \in \{0, 1\}$ (1 = simile, 0 = dissimile), la **contrastive loss** (Hadsell et al., 2006) è:

$$\mathcal{L}_{\text{cont}} = \frac{1}{2N} \sum_{i=1}^N [y_{ij} d_{ij}^2 + (1 - y_{ij}) \max(0, m - d_{ij})^2]$$

dove $d_{ij} = \|f_\theta(x_i) - f_\theta(x_j)\|$ e $m > 0$ è il margine. Coppie positive ($y = 1$) vengono attratte; coppie negative ($y = 0$) vengono spinte ad almeno distanza m .

18.2 Triplet Loss

Usa triple (a, p, n) : anchor, positive (stessa classe), negative (classe diversa):

$$\mathcal{L}_{\text{triplet}} = \frac{1}{N} \sum_{i=1}^N \max(0, \|f(a_i) - f(p_i)\|^2 - \|f(a_i) - f(n_i)\|^2 + \alpha)$$

Il margine $\alpha > 0$ forza che la distanza anchor-negative superi quella anchor-positive di almeno α . La loss è zero quando $d(a, n) - d(a, p) > \alpha$ (tripla "soddisfatta").

Hard negative mining

Per efficienza, si selezionano le triple più difficili: quelle con $d(a, n)$ piccola (negative vicini all'anchor, detti "hard negatives"). Con un mini-batch di N esempi si possono formare $O(N^3)$ triple, ma si usano solo le più informative.

18.3 NT-Xent (SimCLR)

Chen et al. (2020) propongono la NT-Xent (Normalized Temperature-scaled Cross Entropy) per l'apprendimento auto-supervisionato. Dati $2N$ embedding da N coppie di aumentazioni (z_i, z_j) , per ogni coppia positiva (i, j) :

$$\mathcal{L}_{i,j} = -\log \frac{\exp(\text{sim}(z_i, z_j)/\tau)}{\sum_{k=1}^{2N} \mathbf{1}[k \neq i] \exp(\text{sim}(z_i, z_k)/\tau)}$$

dove $\text{sim}(\mathbf{u}, \mathbf{v}) = \mathbf{u}^T \mathbf{v} / (\|\mathbf{u}\| \|\mathbf{v}\|)$ è la similarità coseno e τ è la temperatura. Il denominatore somma su tutti i $2N - 1$ altri esempi nel batch (sia positivi che negativi). Il gradiente rispetto a τ mostra che temperature basse rendono la loss più "selettiva" (penalty alta per falsi negativi vicini).

Teorema di Approssimazione Universale — Costruzione Esplicita

19.1 Bump Functions con Sigmoid

Mostriamo costruttivamente come una rete con un singolo hidden layer può approssimare qualsiasi funzione continua $f : [0, 1] \rightarrow \mathbb{R}$.

Passo 1 — Approssimazione di una funzione a gradino

Con due sigmoid con pesi grandi $w \gg 0$ e bias diversi:

$$H(x; w, b_1, b_2) = \sigma(w(x - b_1)) - \sigma(w(x - b_2)) \xrightarrow{w \rightarrow \infty} \mathbf{1}[b_1 \leq x \leq b_2]$$

Per w grande, questa funzione è quasi una funzione rettangolare (bump): vale ≈ 1 nell'intervallo $[b_1, b_2]$ e ≈ 0 altrove.

Passo 2 — Approssimazione di f con bump functions

Dividiamo $[0, 1]$ in N sottointervalli $I_k = [(k-1)/N, k/N]$ per $k = 1, \dots, N$. Per ogni intervallo, usiamo una bump function scalata per il valore di f al centro dell'intervallo:

$$f_N(x) = \sum_{k=1}^N f\left(\frac{2k-1}{2N}\right) \cdot H\left(x; w, \frac{k-1}{N}, \frac{k}{N}\right)$$

Questa è una rete neurale con $2N$ neuroni nel hidden layer (due per ogni bump).
Per la continuità uniforme di f su $[0, 1]$, $\|f - f_N\|_\infty \rightarrow 0$ per $N \rightarrow \infty$.

19.2 Il Teorema Formale

TEOREMA DI APPROSSIMAZIONE UNIVERSALE (CYBENKO 1989, HORNIK ET AL. 1989)

Sia $\sigma : \mathbb{R} \rightarrow \mathbb{R}$ una funzione di attivazione non-costante, limitata e continua (per es. sigmoid). Allora lo spazio delle funzioni della forma:

$$f(\mathbf{x}) = \sum_{j=1}^N v_j \sigma(\mathbf{w}_j^T \mathbf{x} + b_j)$$

è **denso** in $C([0, 1]^n)$ (funzioni continue su $[0, 1]^n$ con la norma uniforme) per ogni $n \geq 1$.

In altre parole: per ogni $f \in C([0, 1]^n)$ e ogni $\varepsilon > 0$, esiste un MLP con abbastanza neuroni tale che $\sup_{\mathbf{x} \in [0, 1]^n} |f(\mathbf{x}) - \hat{f}(\mathbf{x})| < \varepsilon$.

19.3 Profondità vs. Larghezza

Il teorema garantisce l'esistenza di un'approssimazione con un singolo layer largo.
Ma reti profonde e strette sono spesso più efficienti:

TEOREMA (MONTUFAR ET AL., 2014)

Una rete ReLU di profondità L e larghezza n può rappresentare una funzione lineare a tratti con fino a $\left(\frac{n}{n_0}\right)^{L-1} \binom{n_0}{n_0}$ regioni lineari, dove n_0 è la

dimensione dell'input. Una rete di un singolo layer con lo stesso numero di neuroni ne ha solo $\binom{N}{n_0}$ (crescita polinomiale vs. esponenziale).

Le reti profonde possono rappresentare funzioni con complessità esponenziale nel numero di layer — la profondità è computazionalmente vantaggiosa.

PARTE 20

Teoria della Generalizzazione

20.1 PAC Learning

Il framework **Probably Approximately Correct** (Valiant, 1984) formalizza quando un algoritmo di apprendimento funziona. Un algoritmo PAC-apprende una classe $\mathcal{H}$ se, con N esempi, produce con probabilità $\geq 1 - \delta$ un classificatore con errore $\leq \varepsilon$:

$$P(\mathcal{L}_{\text{test}}(h_N) \leq \varepsilon) \geq 1 - \delta$$

Il numero di esempi necessari dipende dalla complessità della classe $\mathcal{H}$ (solitamente misurata dalla VC dimension) e dai parametri ε, δ :

$$N \geq \frac{1}{\varepsilon} \left(d_{\text{VC}} \log \frac{2}{\varepsilon} + \log \frac{2}{\delta} \right)$$

20.2 VC Dimension

La **VC dimension** (Vapnik-Chervonenkis) di una classe di ipotesi $\mathcal{H}$ è il massimo numero di punti che $\mathcal{H}$ può "shatterare" — classificare in tutti i 2^N modi possibili:

$$d_{VC}(\mathcal{H}) = \max\{N : \exists\{x_1, \dots, x_N\} \text{ shatteabile da } \mathcal{H}\}$$

Per un classificatore lineare in $\mathbb{R}^d$: $d_{VC} = d + 1$ (può shatterare $d + 1$ punti in posizione generica). Per un MLP con W parametri: $d_{VC} = O(W \log W)$.

Teorema fondamentale — bound sull'errore di generalizzazione

TEOREMA (VAPNIK-CHERVONENKIS)

Con probabilità $\geq 1 - \delta$ su un campione di N esempi, per ogni $h \in \mathcal{H}$:

$$\mathcal{L}_{\text{test}}(h) \leq \mathcal{L}_{\text{train}}(h) + O\left(\sqrt{\frac{d_{VC} \log N + \log(1/\delta)}{N}}\right)$$

Il secondo termine è il **gap di generalizzazione**: cresce con la complessità del modello (d_{VC}) e decresce con il numero di esempi (N).

20.3 Rademacher Complexity

Un'alternativa alla VC dimension che dipende dai dati specifici. La complessità di Rademacher di una classe $\mathcal{F}$ su un campione $S = \{x_1, \dots, x_N\}$ è:

$$\hat{\mathcal{R}}_S(\mathcal{F}) = \mathbb{E}_{\sigma} \left[\sup_{f \in \mathcal{F}} \frac{1}{N} \sum_{i=1}^N \sigma_i f(x_i) \right]$$

dove $\sigma_i \sim \text{Uniform}(\{-1, +1\})$ sono variabili di Rademacher. Misura quanto bene le funzioni in $\mathcal{F}$ riescono a correlare con rumore casuale. Più alta è la complessità, più il modello può overfittare.

Bound di generalizzazione via Rademacher

Con probabilità $\geq 1 - \delta$:

$$\mathcal{L}_{\text{test}}(f) \leq \mathcal{L}_{\text{train}}(f) + 2\hat{\mathcal{R}}_S(\mathcal{F}) + 3\sqrt{\frac{\log(2/\delta)}{2N}}$$

20.4 Double Descent

Il fenomeno del **double descent** (Belkin et al., 2019) sfida la visione classica bias-varianza. All'aumentare del numero di parametri p :

1. **Regime underparametrizzato** ($p < N$): l'errore di test decresce man mano che il modello diventa più espressivo (comportamento classico)
2. **Picco di interpolazione** ($p \approx N$): il modello inizia a memorizzare i dati, l'errore di test esplode
3. **Regime overparametrizzato** ($p \gg N$): il modello è così sovradimensionato che la soluzione con norma minima generalizza bene — l'errore di test decresce di nuovo

Questo spiega perché le grandi reti neurali (con $p \gg N$) generalizzano spesso meglio di reti più piccole, sfidando la saggezza convenzionale del machine learning.

PARTE 21

Ottimizzatori Avanzati

21.1 AdaGrad

AdaGrad (Duchi et al., 2011) accumula la somma dei quadrati dei gradienti passati per adattare il learning rate per parametro:

$$G_{t,i} = G_{t-1,i} + g_{t,i}^2 \quad (\text{somma cumulativa dei quadrati})$$

$$\theta_{t,i} = \theta_{t-1,i} - \frac{\alpha}{\sqrt{G_{t,i} + \epsilon}} g_{t,i}$$

Parametri con gradienti storicamente grandi ricevono passi più piccoli; quelli con gradienti piccoli ricevono passi più grandi. Problema: $G_{t,i}$ cresce monotonamente, quindi il learning rate effettivo $\alpha/\sqrt{G_{t,i}}$ tende a zero — l'apprendimento si ferma.

21.2 RMSprop

Hinton (2012) propose di usare una media mobile esponenziale invece della somma cumulativa, evitando il problema di AdaGrad:

$$v_t = \rho v_{t-1} + (1 - \rho) g_t^2 \quad (\rho \approx 0.9)$$

$$\theta_t = \theta_{t-1} - \frac{\alpha}{\sqrt{v_t + \varepsilon}} g_t$$

v_t è la media mobile del quadrato del gradiente — si "dimentica" i gradienti vecchi. RMSprop è particolarmente efficace per reti ricorrenti.

Connessione Adam-RMSprop

Adam può essere visto come RMSprop con momentum sul gradiente: il secondo momento v_t di Adam è identico a RMSprop, mentre il primo momento m_t aggiunge la direzione del gradiente smoothed. Le correzioni del bias distinguono Adam da RMSprop all'inizio del training.

21.3 LAMB (Large Batch Optimizer)

LAMB (You et al., 2019) è progettato per training con batch molto grandi (es. 32,768 esempi). Scala l'aggiornamento di Adam per la norma del layer:

$$\theta_t = \theta_{t-1} - \alpha \cdot \frac{\|\theta_{t-1}\|}{\|u_t\|} \cdot u_t$$

dove $u_t = \hat{m}_t / (\sqrt{\hat{v}_t + \varepsilon}) + \lambda \theta_{t-1}$ è l'aggiornamento Adam standard con weight decay. Il rapporto $\|\theta\|/\|u\|$ adatta l'ampiezza del passo alla norma del layer — layer con pesi grandi fanno passi più grandi.

21.4 Lookahead

Lookahead (Zhang et al., 2019) avvolge qualsiasi ottimizzatore base con un meccanismo di "lookahead": mantiene pesi "lenti" ϕ che vengono aggiornati periodicamente verso i pesi "veloci" θ :

ALGORITMO LOOKAHEAD

Dato un ottimizzatore base (es. Adam) e iperparametri k (frequenza di sync) e α (step size lento):

- Per ogni k passi: aggiorna θ con l'ottimizzatore base (passi "veloci")
- Ogni k passi: $\phi_{t+1} = \phi_t + \alpha(\theta_{t+k} - \phi_t)$ (passo "lento")
- Reset: $\theta \leftarrow \phi_{t+1}$ (riparti dai pesi lenti)

I pesi lenti esplorano l'intorno dei pesi veloci e si "aggiornano" nella direzione media — questo riduce la varianza e migliora la stabilità, specialmente per learning rate alti.

PARTE 22

Architetture Moderne dei Grandi Modelli

22.1 Mixture of Experts (MoE)

Invece di passare ogni token attraverso la stessa FFN, MoE usa E reti "esperte" e un router che seleziona K di esse per ogni token (sparse MoE):

$$y = \sum_{i=1}^E G(\mathbf{x})_i \cdot E_i(\mathbf{x})$$

dove $G(\mathbf{x}) = \text{TopK}(\text{softmax}(\mathbf{x}\mathbf{W}_g), K)$ — vengono attivati solo i K esperti con peso più alto, gli altri ricevono peso 0.

Auxiliary load balancing loss

Senza vincoli, il router tende a selezionare sempre gli stessi esperti (collasso). Si aggiunge una loss ausiliaria che incoraggia il bilanciamento:

$$\mathcal{L}_{\text{aux}} = \alpha \sum_{i=1}^E f_i \cdot P_i$$

dove $f_i = \frac{1}{T} \sum_t \mathbf{1}[\text{token } t \text{ mandato all'esperto } i]$ è la frazione di token all'esperto i e $P_i = \frac{1}{T} \sum_t G(\mathbf{x}_t)_i$ è il peso medio dell'esperto i . Minimizzare il prodotto $f_i \cdot P_i$ scoraggia la concentrazione.

22.2 Flash Attention

Il collo di bottiglia dell'attention standard è il calcolo e la memorizzazione della matrice $\mathbf{QK}^T \in \mathbb{R}^{n \times n}$, con complessità $O(n^2)$ in memoria.

Flash Attention (Dao et al., 2022) ricalcola l'attenzione in blocchi (tiling) per evitare di materializzare l'intera matrice:

Memory: $O(n^2) \rightarrow O(n)$, Compute: $O(n^2d)$ (uguale, ma con $2-4 \times$

L'idea chiave è la decomposizione online della softmax. Per la riga i di $\mathbf{A}$, si accumula il numeratore $\sum_j e^{s_{ij}} v_j$ e il denominatore $\sum_j e^{s_{ij}}$ incrementalmente, tenendo traccia del massimo locale per la stabilità numerica:

$$m_i^{(\text{new})} = \max(m_i^{(\text{old})}, \max_j s_{ij}^{(\text{block})})$$

$$l_i^{(\text{new})} = e^{m_i^{(\text{old})} - m_i^{(\text{new})}} l_i^{(\text{old})} + \sum_j e^{s_{ij} - m_i^{(\text{new})}}$$

$$o_i^{(\text{new})} = \frac{e^{m_i^{(\text{old})} - m_i^{(\text{new})}} l_i^{(\text{old})}}{l_i^{(\text{new})}} o_i^{(\text{old})} + \frac{1}{l_i^{(\text{new})}} \sum_j e^{s_{ij} - m_i^{(\text{new})}} v_j$$

22.3 RoPE — Derivazione Completa della Proprietà di Posizione Relativa

RoPE (Su et al., 2021) incorpora la posizione direttamente nel prodotto scalare Q-K. Vogliamo che il prodotto $q_m^T k_n$ dipenda solo dalla differenza $(m - n)$, non dalle posizioni assolute m e n .

Per una singola coppia di dimensioni, definiamo la trasformazione:

$$f_q(q, m) = q e^{im\theta} \quad (\text{rotazione nel piano complesso})$$

$$f_k(k, n) = k e^{in\theta}$$

Il prodotto scalare diventa:

$$\text{Re}[f_q(q, m) \overline{f_k(k, n)}] = \text{Re}[q e^{im\theta} \overline{k e^{in\theta}}] = \text{Re}[q \bar{k} e^{i(m-n)\theta}]$$

Dipende solo da $m - n$! In forma reale, per la coppia di dimensioni $(2i, 2i + 1)$ con $\theta_i = 1/10000^{2i/d}$:

$$\mathbf{R}(m, i) = \begin{pmatrix} \cos m\theta_i & -\sin m\theta_i \\ \sin m\theta_i & \cos m\theta_i \end{pmatrix}$$

$$q_{2i:2i+1}^{(m)} = \mathbf{R}(m, i) q_{2i:2i+1}, \quad k_{2i:2i+1}^{(n)} = \mathbf{R}(n, i) k_{2i:2i+1}$$

Il prodotto scalare della coppia i è:

$$(q_{2i:2i+1}^{(m)})^T k_{2i:2i+1}^{(n)} = q_{2i:2i+1}^T \mathbf{R}(m, i)^T \mathbf{R}(n, i) k_{2i:2i+1} = q_{2i:2i+1}^T \mathbf{R}(n - m, i) k_{2i:2i+1}$$

poiché $\mathbf{R}(m)^T \mathbf{R}(n) = \mathbf{R}(-m) \mathbf{R}(n) = \mathbf{R}(n - m)$. L'intero prodotto scalare $Q \cdot K$ è quindi funzione di $n - m$.

PARTE 23

Funzioni di Attivazione Avanzate

23.1 ELU (Exponential Linear Unit)

$$\text{ELU}(z) = \begin{cases} z & z > 0 \\ \alpha(e^z - 1) & z \leq 0 \end{cases} \quad (\alpha > 0)$$

A differenza di ReLU, ELU produce output negativi per input negativi — questo porta la media delle attivazioni più vicino a zero, migliorando la propagazione dei

gradienti. La derivata è continua in $z = 0$:

$$\text{ELU}'(z) = \begin{cases} 1 & z > 0 \\ \text{ELU}(z) + \alpha & z \leq 0 \end{cases}$$

23.2 Mish

Mish (Misra, 2019) è definita come z moltiplicato per la tangente iperbolica della softplus di z :

$$\text{Mish}(z) = z \cdot \tanh(\ln(1 + e^z)) = z \cdot \tanh(\text{softplus}(z))$$

La softplus $\text{sp}(z) = \ln(1 + e^z)$ è un'approssimazione morbida della ReLU. Mish è non-monotona (decresce leggermente per z molto negativi) e autoregolarizzante.

La derivata:

$$\text{Mish}'(z) = \tanh(\text{sp}(z)) + z \cdot \text{sech}^2(\text{sp}(z)) \cdot \sigma(z)$$

23.3 GLU e Varianti (SwiGLU, GeGLU)

Le **Gated Linear Units** (Dauphin et al., 2017) usano due proiezioni lineari e una funzione di gate:

$$\text{GLU}(\mathbf{x}, \mathbf{W}, \mathbf{V}) = (\mathbf{x}\mathbf{W}) \odot \sigma(\mathbf{x}\mathbf{V})$$

La seconda proiezione $\sigma(\mathbf{xV})$ agisce da gate: decide quali componenti passare.
Varianti moderne sostituiscono σ con altre attivazioni:

Nome	Formula	Usato in
GLU	$(\mathbf{xW}) \odot \sigma(\mathbf{xV})$	Originale
ReGLU	$(\mathbf{xW}) \odot \text{ReLU}(\mathbf{xV})$	
GeGLU	$(\mathbf{xW}) \odot \text{GELU}(\mathbf{xV})$	PaLM, T5
SwiGLU	$(\mathbf{xW}) \odot \text{SiLU}(\mathbf{xV})$	LLaMA, Mistral, Gemma

Derivata di SwiGLU rispetto a $\mathbf{x}$

Per la regola del prodotto e della catena, con $a = \mathbf{xW}$ e $b = \text{SiLU}(\mathbf{xV})$:

$$\frac{\partial}{\partial \mathbf{x}} (a \odot b) = b \odot \mathbf{W}^T + a \odot \frac{\partial b}{\partial \mathbf{x}}$$
$$\frac{\partial \text{SiLU}(\mathbf{xV})}{\partial \mathbf{x}} = \text{SiLU}'(\mathbf{xV}) \odot \mathbf{V}^T$$

PARTE 24

Attenzione Avanzata: Cross-Attention, Sparse, Linear

24.1 Cross-Attention

Nella self-attention, Query, Key e Value vengono tutte dallo stesso input $\mathbf{X}$. Nella **cross-attention**, le Query vengono da una sequenza $\mathbf{X}$ e le Key/Value da un'altra sequenza $\mathbf{C}$ (il "contesto"):

$$\mathbf{Q} = \mathbf{X}\mathbf{W}^Q, \quad \mathbf{K} = \mathbf{C}\mathbf{W}^K, \quad \mathbf{V} = \mathbf{C}\mathbf{W}^V$$
$$\text{CrossAttn}(\mathbf{X}, \mathbf{C}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}$$

Ogni posizione di $\mathbf{X}$ "chiede" (Q) informazioni al contesto $\mathbf{C}$, che risponde con le sue Key e Value. Questo è il meccanismo usato nel decoder del Transformer originale per "guardare" l'output dell'encoder.

24.2 Sparse Attention

Per sequenze lunghe ($n \gg 1000$), l'attenzione $O(n^2)$ diventa proibitiva. La **sparse attention** limita ogni token ad attendere solo un sottoinsieme di posizioni. Uno schema comune (Sparse Transformer, Child et al., 2019):

- **Local attention:** ogni token attende ai w token precedenti (finestra locale)
- **Strided attention:** ogni token attende ai token a distanza multipla di s (stride)

Combinando i due schemi, ogni token è connesso a tutti gli altri in $O(\log n)$ passi di attenzione, mantenendo complessità $O(n\sqrt{n})$ invece di $O(n^2)$.

24.3 Linear Attention

La complessità quadratica viene dalla softmax, che richiede di calcolare $\mathbf{Q}\mathbf{K}^T$ prima di moltiplicare per $\mathbf{V}$. Se approssimiamo la softmax con una funzione kernel

$$k(\mathbf{q}, \mathbf{k}) = \phi(\mathbf{q})^T \phi(\mathbf{k}):$$

$$\text{Attn}(\mathbf{q}_i, \mathbf{K}, \mathbf{V})_i = \frac{\sum_j \phi(\mathbf{q}_i)^T \phi(\mathbf{k}_j) \mathbf{v}_j}{\sum_j \phi(\mathbf{q}_i)^T \phi(\mathbf{k}_j)} = \frac{\phi(\mathbf{q}_i)^T \sum_j \phi(\mathbf{k}_j) \mathbf{v}_j^T}{\phi(\mathbf{q}_i)^T \sum_j \phi(\mathbf{k}_j)}$$

Si calcolano prima $\sum_j \phi(\mathbf{k}_j) \mathbf{v}_j^T \in \mathbb{R}^{d \times d}$ e $\sum_j \phi(\mathbf{k}_j) \in \mathbb{R}^d$ (una volta per tutti), poi si calcola il prodotto per ogni $\mathbf{q}_i$. Complessità totale: $O(nd^2)$ invece di $O(n^2d)$.

24.4 KV Cache nell'Inferenza

Durante la generazione autoregressiva, ad ogni passo t si genera un nuovo token ma si mantengono le Key e Value di tutti i token precedenti in una cache. Senza cache: ogni passo richiederebbe $O(t^2)$ per ricalcolare tutta l'attenzione. Con cache: solo $O(t)$ per calcolare le nuove K,V e fare il prodotto con le query del nuovo token.

$$\text{Memoria KV-cache} = 2 \cdot L \cdot n_{\text{heads}} \cdot d_{\text{head}} \cdot t_{\text{max}} \cdot \text{sizeof}(\text{dtype})$$

Per un modello con $L = 32$ layer, $n_{\text{heads}} = 32$, $d_{\text{head}} = 128$, sequenza di 4096 token in float16 (2 byte): $2 \times 32 \times 32 \times 128 \times 4096 \times 2 = 2$ GB. Per questo GQA (ridurre le teste K,V) è fondamentale per l'inferenza efficiente.

PARTE 25

Vision Transformer (ViT)

Come applicare il Transformer alle immagini: patch embedding, class token, positional encoding 2D, e l'analisi dei meccanismi di attenzione visiva.

25.1 Patch Embedding — Da Immagine a Sequenza

Un ViT (Dosovitskiy et al., 2021) converte un'immagine $\mathbf{x} \in \mathbb{R}^{H \times W \times C}$ in una sequenza di token suddividendo in patch $P \times P$ non sovrapposte:

$$N = \frac{HW}{P^2} \quad (\text{numero di patch}), \quad \text{ogni patch} \in \mathbb{R}^{P^2 C}$$

Ogni patch viene linearizzata e proiettata in $\mathbb{R}^d$ tramite una matrice di embedding $\mathbf{E} \in \mathbb{R}^{(P^2 C) \times d}$. Un **class token** apprendibile $\mathbf{x}_{\text{cls}} \in \mathbb{R}^d$ viene preposto alla sequenza, portando il totale a $N + 1$ token:

$$\mathbf{Z}_0 = [\mathbf{x}_{\text{cls}}; \mathbf{x}_p^1 \mathbf{E}; \mathbf{x}_p^2 \mathbf{E}; \dots; \mathbf{x}_p^N \mathbf{E}] + \mathbf{E}_{\text{pos}}$$

dove $\mathbf{E}_{\text{pos}} \in \mathbb{R}^{(N+1) \times d}$ sono i positional embedding 1D appresi. L'output del class token all'ultimo layer viene usato per la classificazione.

Costo computazionale rispetto alle CNN

Con patch $P = 16$ su immagini 224×224 : $N = 196$ token. Ogni layer Transformer ha complessità $O(N^2 d)$ per l'attenzione. Per $N = 196$ e $d = 768$: l'attenzione costa $196^2 \times 768 \approx 30M$ operazioni per layer — comparabile a una ResNet-50.

25.2 Positional Encoding 2D

Il PE appreso 1D non cattura la struttura 2D dell'immagine. Alternative:

2D sinusoidale: per la patch alla riga r , colonna c (con $r, c \in \{0, \dots, \sqrt{N} - 1\}$):

$$\begin{aligned}\mathbf{PE}(r, c, 2i) &= \sin\left(\frac{r}{10000^{4i/d}}\right), & \mathbf{PE}(r, c, 2i + 1) &= \cos\left(\frac{r}{10000^{4i/d}}\right) \\ \mathbf{PE}(r, c, d/2 + 2i) &= \sin\left(\frac{c}{10000^{4i/d}}\right), & \dots\end{aligned}$$

La prima metà delle dimensioni codifica la riga, la seconda la colonna. Questo è il PE usato in DeiT e molti ViT successivi.

25.3 Analisi dei Pattern di Attenzione Visiva

Nei ViT addestrati, i pattern di attenzione sviluppano proprietà interessanti. Alla profondità del layer ℓ , il **rollout dell'attenzione** misura il flusso di informazione complessivo dall'input al class token:

$$\hat{\mathbf{A}}^{(\ell)} = 0.5 \mathbf{A}^{(\ell)} + 0.5 \mathbf{I} \quad (\text{aggiunge identità per la residual connection})$$

$$\mathbf{R}^{(L)} = \hat{\mathbf{A}}^{(1)} \cdot \hat{\mathbf{A}}^{(2)} \dots \hat{\mathbf{A}}^{(L)}$$

La riga del class token in $\mathbf{R}^{(L)}$ indica quali patch contribuiscono maggiormente alla predizione finale — produce una heatmap di saliency senza bisogno di backpropagation.

BERT e Masked Language Modeling

26.1 Masked Language Modeling (MLM)

BERT (Devlin et al., 2019) usa un obiettivo di preaddestramento bidirezionale: mascherare il 15% dei token e predirli. Per ogni token selezionato:

- 80% delle volte: sostituito con `[MASK]`
- 10% delle volte: sostituito con un token casuale
- 10% delle volte: lasciato invariato

La loss MLM è la cross-entropy solo sui token mascherati:

$$\mathcal{L}_{\text{MLM}} = -\frac{1}{|\mathcal{M}|} \sum_{i \in \mathcal{M}} \log P_{\theta}(x_i \mid \mathbf{x}_{\setminus \mathcal{M}})$$

dove $\mathcal{M}$ è l'insieme degli indici mascherati e $\mathbf{x}_{\setminus \mathcal{M}}$ è la sequenza con i token mascherati.

Differenza con GPT (autoregressivo)

GPT usa attenzione causale: ogni token vede solo i precedenti. BERT usa attenzione bidirezionale: ogni token vede tutti gli altri (esclusi i mascherati durante il training). Questo rende BERT più adatto per task di comprensione (classificazione, NER, QA) ma incapace di generazione autoregressiva.

26.2 Fine-Tuning: Adattamento ai Task Downstream

Il fine-tuning di BERT aggiunge un head specifico al task sull'output del [CLS] token (o di tutti i token per task a livello di token):

Classificazione di sequenza: $\hat{y} = \text{softmax}(\mathbf{W}_C \mathbf{h}_{\text{CLS}} + \mathbf{b}_C)$

Token classification (NER): $\hat{y}_i = \text{softmax}(\mathbf{W}_T \mathbf{h}_i + \mathbf{b}_T)$ per ogni token i

Question Answering: predice start e end span: $P_{\text{start}}(i) = \text{softmax}(\mathbf{s}^T \mathbf{h}_i)$,
 $P_{\text{end}}(i) = \text{softmax}(\mathbf{e}^T \mathbf{h}_i)$

La loss di fine-tuning è la somma delle loss specifiche dei task. Tutti i parametri (inclusi quelli di BERT) vengono aggiornati, con LR molto più basso ($2-5 \times 10^{-5}$) rispetto al preaddestramento.

PARTE 27

State Space Models (SSM) e Mamba

Un'alternativa al Transformer basata su sistemi dinamici lineari: complessità $O(n)$ nella lunghezza della sequenza.

27.1 Il Modello SSM Continuo

Un SSM mappa una sequenza di input $u(t) \in \mathbb{R}$ in un output $y(t) \in \mathbb{R}$ tramite uno stato latente $\mathbf{h}(t) \in \mathbb{R}^N$:

SSM — FORMA CONTINUA

$$\dot{\mathbf{h}}(t) = \mathbf{A}\mathbf{h}(t) + \mathbf{B}u(t)$$

$$y(t) = \mathbf{C}\mathbf{h}(t) + Du(t)$$

dove $\mathbf{A} \in \mathbb{R}^{N \times N}$ è la matrice di stato, $\mathbf{B} \in \mathbb{R}^{N \times 1}$, $\mathbf{C} \in \mathbb{R}^{1 \times N}$ e $D \in \mathbb{R}$ è il skip connection.

27.2 Discretizzazione

Per applicare l'SSM a sequenze discrete, si discretizza con step size Δ (apprendibile). Con il metodo **Zero-Order Hold (ZOH)**:

- 1 Soluzione esatta di $\dot{\mathbf{h}} = \mathbf{A}\mathbf{h} + \mathbf{B}u$ su $[t, t + \Delta]$:

$$\mathbf{h}(t + \Delta) = e^{\mathbf{A}\Delta}\mathbf{h}(t) + \int_0^\Delta e^{\mathbf{A}(\Delta-\tau)}\mathbf{B} d\tau \cdot u(t)$$

Soluzione della EDO lineare con input costante (ZOH)

- 2 $\bar{\mathbf{A}} = e^{\mathbf{A}\Delta}$, $\bar{\mathbf{B}} = (\mathbf{A})^{-1}(e^{\mathbf{A}\Delta} - \mathbf{I})\mathbf{B} \approx \Delta\mathbf{B}$ per Δ piccolo

Calcolo dell'integrale e definizione delle matrici discrete

- 3 Forma ricorrente discreta:

$$\mathbf{h}_k = \bar{\mathbf{A}}\mathbf{h}_{k-1} + \bar{\mathbf{B}}u_k, \quad y_k = \mathbf{C}\mathbf{h}_k + Du_k$$

Valutazione del punto discreto $t_k = k\Delta$

$$\bar{\mathbf{A}} = e^{\mathbf{A}\Delta}, \quad \bar{\mathbf{B}} = (\mathbf{A})^{-1}(e^{\mathbf{A}\Delta} - \mathbf{I})\mathbf{B}, \quad y_k = \mathbf{C}\bar{\mathbf{A}}^k\mathbf{h}_0 + \sum_{j=0}^k \mathbf{C}\bar{\mathbf{A}}^{k-j}\bar{\mathbf{B}}u_j$$

27.3 Equivalenza con la Convoluzione

L'SSM discreto può essere eseguito come una convoluzione 1D! Definendo il kernel convoluzionale:

$$\bar{\mathbf{K}} = (\mathbf{C}\bar{\mathbf{B}}, \mathbf{C}\bar{\mathbf{A}}\bar{\mathbf{B}}, \mathbf{C}\bar{\mathbf{A}}^2\bar{\mathbf{B}}, \dots, \mathbf{C}\bar{\mathbf{A}}^{L-1}\bar{\mathbf{B}}) \in \mathbb{R}^L$$

$$\mathbf{y} = \mathbf{u} * \bar{\mathbf{K}}$$

La convoluzione può essere computata efficientemente tramite FFT in $O(L \log L)$. Questo permette parallelizzazione completa durante il training (come il Transformer), mantenendo la ricorrenza lineare durante l'inferenza ($O(N)$ memoria, $O(1)$ per token).

27.4 Mamba: SSM Selettivo

Il problema degli SSM classici: le matrici $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ non dipendono dall'input — l'SSM non può "selezionare" quali informazioni propagare. Mamba (Gu e Dao, 2023) rende i parametri dipendenti dall'input:

$$\mathbf{B}_k = s_B(u_k), \quad \mathbf{C}_k = s_C(u_k), \quad \Delta_k = \text{softplus}(s_\Delta(u_k))$$

dove s_B, s_C, s_Δ sono proiezioni lineari apprese. Ora il kernel varia per ogni posizione — impossibile pre-computarlo. Mamba usa un algoritmo di **parallel scan** (scan parallelo) per calcolare la ricorrenza in $O(L)$ tempo su GPU mantenendo la capacità selettiva.

PARTE 28

Normalizing Flows

Modelli generativi esatti: mappare una distribuzione semplice (gaussiana) in una complessa tramite trasformazioni invertibili, con log-likelihood calcolabile

28.1 Change of Variables Formula

Sia $\mathbf{z} \sim p_Z(\mathbf{z})$ (es. gaussiana standard) e $\mathbf{x} = f(\mathbf{z})$ con f invertibile e differenziabile. La distribuzione di $\mathbf{x}$ è:

TEOREMA DEL CAMBIO DI VARIABILE

$$p_X(\mathbf{x}) = p_Z(f^{-1}(\mathbf{x})) |\det \mathbf{J}_{f^{-1}}(\mathbf{x})| = p_Z(f^{-1}(\mathbf{x})) |\det \mathbf{J}_f(f^{-1}(\mathbf{x}))|^{-1}$$

In log-spazio:

$$\log p_X(\mathbf{x}) = \log p_Z(f^{-1}(\mathbf{x})) - \log |\det \mathbf{J}_f(\mathbf{z})|$$

dove $\mathbf{J}_f = \partial f / \partial \mathbf{z}$ è la matrice Jacobiana di f .

Il termine $\log |\det \mathbf{J}_f|$ è il **log-determinante del Jacobiano**: misura quanto la trasformazione "espande" o "comprime" il volume locale. Per una trasformazione generica $d \times d$ il determinante costa $O(d^3)$ — occorre progettare f con Jacobiano a struttura speciale.

28.2 Composizione di Flow

Un normalizing flow è una composizione di K trasformazioni invertibili

$$f = f_K \circ \dots \circ f_1:$$

$$\mathbf{z}_0 \sim p_0, \quad \mathbf{z}_k = f_k(\mathbf{z}_{k-1}), \quad k = 1, \dots, K$$

$$\log p_X(\mathbf{x}) = \log p_0(\mathbf{z}_0) - \sum_{k=1}^K \log |\det \mathbf{J}_{f_k}(\mathbf{z}_{k-1})|$$

28.3 Coupling Layers (RealNVP)

RealNVP (Dinh et al., 2017) usa coupling layers con Jacobiano triangolare. Divide $\mathbf{z}$ in due parti $(\mathbf{z}_{1:d}, \mathbf{z}_{d+1:D})$:

$$\begin{aligned}\mathbf{x}_{1:d} &= \mathbf{z}_{1:d} \\ \mathbf{x}_{d+1:D} &= \mathbf{z}_{d+1:D} \odot \exp(s(\mathbf{z}_{1:d})) + t(\mathbf{z}_{1:d})\end{aligned}$$

dove s (scale) e t (translate) sono reti neurali arbitrarie. La prima metà non viene trasformata; la seconda viene scalata e traslata in modo dipendente dalla prima.

Il Jacobiano di questa trasformazione è **triangolare a blocchi**:

$$\mathbf{J}_f = \begin{pmatrix} \mathbf{I}_d & \mathbf{0} \\ \partial \mathbf{x}_{d+1:D} / \partial \mathbf{z}_{1:d} & \text{diag}(\exp(s(\mathbf{z}_{1:d}))) \end{pmatrix}$$

Il determinante di una matrice triangolare a blocchi è il prodotto dei determinanti dei blocchi diagonali:

$$\log |\det \mathbf{J}_f| = \sum_{j=d+1}^D s_j(\mathbf{z}_{1:d})$$

Costo: $O(D)$ invece di $O(D^3)$! L'inversione è analoga: $\mathbf{z}_{1:d} = \mathbf{x}_{1:d}$,

$$\mathbf{z}_{d+1:D} = (\mathbf{x}_{d+1:D} - t(\mathbf{x}_{1:d})) \odot \exp(-s(\mathbf{x}_{1:d})).$$

Energy-Based Models (EBM)

29.1 Il Modello

Un EBM definisce una distribuzione tramite una funzione di energia

$E_\theta(\mathbf{x}) : \mathbb{R}^d \rightarrow \mathbb{R}$ (bassa energia = alta probabilità):

$$p_\theta(\mathbf{x}) = \frac{\exp(-E_\theta(\mathbf{x}))}{Z_\theta}, \quad Z_\theta = \int \exp(-E_\theta(\mathbf{x})) d\mathbf{x}$$

Z_θ è la **partition function** — intrattabile per dati continui. Il gradiente della log-likelihood rispetto a θ è:

- 1 $\nabla_\theta \log p_\theta(\mathbf{x}) = -\nabla_\theta E_\theta(\mathbf{x}) - \nabla_\theta \log Z_\theta$

Derivata di $-E_\theta - \log Z_\theta$

- 2 $\nabla_\theta \log Z_\theta = \frac{\nabla_\theta Z_\theta}{Z_\theta} = \frac{\int (-\nabla_\theta E_\theta(\mathbf{x}')) e^{-E_\theta(\mathbf{x}')} d\mathbf{x}'}{Z_\theta} = -\mathbb{E}_{p_\theta}[\nabla_\theta E_\theta(\mathbf{x}')]]$

Derivata sotto segno di integrale e definizione di valore atteso

- 3 $\nabla_\theta \log p_\theta(\mathbf{x}) = -\nabla_\theta E_\theta(\mathbf{x}) + \mathbb{E}_{p_\theta}[\nabla_\theta E_\theta(\mathbf{x}')]]$

Sostituzione

$$\nabla_\theta \mathcal{L} = \mathbb{E}_{p_{\text{data}}}[\nabla_\theta E_\theta(\mathbf{x})] - \mathbb{E}_{p_\theta}[\nabla_\theta E_\theta(\mathbf{x}')]]$$

Il gradiente è la differenza tra l'energia media sui dati reali e l'energia media su campioni dal modello — si vuole che il modello assegni energia bassa ai dati reali

e alta ai campioni.

29.2 Contrastive Divergence

Il problema: campionare da p_θ richiede MCMC (Markov Chain Monte Carlo), lento a convergere. Hinton (2002) propose la **Contrastive Divergence (CD- k)**: invece di campionare fino alla convergenza, si fanno solo k passi di Gibbs partendo da un dato reale:

$$\begin{aligned}\tilde{\mathbf{x}} &= \text{MCMC}^{(k)}(\mathbf{x}_{\text{data}}, p_\theta) \quad (k \text{ passi di Gibbs}) \\ \nabla_\theta \mathcal{L}_{\text{CD-}k} &\approx \nabla_\theta E_\theta(\mathbf{x}_{\text{data}}) - \nabla_\theta E_\theta(\tilde{\mathbf{x}})\end{aligned}$$

Per $k = 1$ (CD-1): partendo da un dato reale, si fa un solo passo Gibbs per ottenere il "fantasy particle" $\tilde{\mathbf{x}}$.

29.3 Score Matching

Un approccio alternativo che evita la partition function. Lo **score** di una distribuzione è $\nabla_{\mathbf{x}} \log p(\mathbf{x})$. Per un EBM: $\nabla_{\mathbf{x}} \log p_\theta(\mathbf{x}) = -\nabla_{\mathbf{x}} E_\theta(\mathbf{x})$ (la partition function sparisce!). Score matching minimizza:

$$\mathcal{L}_{\text{SM}} = \mathbb{E}_{p_{\text{data}}} \left[\frac{1}{2} \|\nabla_{\mathbf{x}} \log p_\theta(\mathbf{x}) - \nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})\|^2 \right]$$

Hyvarinen (2005) mostrò che, con integrazione per parti, questa loss si può calcolare senza conoscere p_{data} :

$$\mathcal{L}_{\text{SM}} = \mathbb{E}_{p_{\text{data}}} \left[\text{tr}(\nabla_{\mathbf{x}}^2 \log p_{\theta}(\mathbf{x})) + \frac{1}{2} \|\nabla_{\mathbf{x}} \log p_{\theta}(\mathbf{x})\|^2 \right] + C$$

I diffusion model sono strettamente connessi allo score matching: predire il rumore aggiunto è equivalente a predire $-\nabla_{\mathbf{x}} \log q(\mathbf{x}_t|\mathbf{x}_0)$ (lo score della distribuzione perturbata).

PARTE 30

Self-Supervised Learning

Imparare rappresentazioni senza label tramite pretext task: BYOL, DINO, MAE.

30.1 BYOL — Bootstrap Your Own Latent

BYOL (Grill et al., 2020) apprende senza esempi negativi usando due reti asimmetriche:

- **Online network** f_{θ} : predictor + projector, aggiornata con backprop
- **Target network** f_{ξ} : senza predictor, aggiornata con EMA: $\xi \leftarrow \tau\xi + (1 - \tau)\theta$

$$\mathcal{L}_{\text{BYOL}} = \left\| \bar{q}_{\theta}(\mathbf{z}^{\text{online}}) - \bar{\mathbf{z}}^{\text{target}} \right\|^2 = 2 - 2 \frac{q_{\theta}(\mathbf{z}^{\text{online}})^T \mathbf{z}^{\text{target}}}{\|q_{\theta}(\mathbf{z}^{\text{online}})\| \|\mathbf{z}^{\text{target}}\|}$$

dove $\bar{\cdot}$ indica la normalizzazione L2. La stop-gradient è applicata alla target network ($\mathbf{z}^{\text{target}}$ non propaga gradiente). Il collapse (tutte le rappresentazioni uguali) viene evitato dall'asimmetria online/target e dall'EMA.

30.2 DINO

DINO (Caron et al., 2021) usa una struttura simile a BYOL ma con distillazione di distribuzione. Le due viste v_1, v_2 di un'immagine vengono processate dai due rami:

$$\mathcal{L}_{\text{DINO}} = - \sum_{\substack{v_1 \in V^g, \\ v_2 \in V \setminus v_1}} \sum_{k=1}^K p_t^{(k)}(v_1) \log p_s^{(k)}(v_2)$$

dove p_s e p_t sono le distribuzioni softmax degli output degli studente e teacher, e V^g sono le "global views" (crop grandi, $> 50\%$ dell'immagine). Il **centering** del teacher (sottrae la media running) previene il collapse.

30.3 MAE — Masked Autoencoder

MAE (He et al., 2022) applica la masked prediction alle immagini in modo asimmetrico:

- **Encoder** pesante: processa solo il 25% delle patch visibili (non mascherate)
- **Decoder** leggero: ricostruisce le patch mascherate dalle patch visibili + mask token

$$\mathcal{L}_{\text{MAE}} = \frac{1}{|\mathcal{M}|} \sum_{(i,j) \in \mathcal{M}} \|\mathbf{x}_{ij}^{\text{norm}} - \hat{\mathbf{x}}_{ij}\|^2$$

dove la target è l'immagine originale normalizzata per patch. Mascherare il 75% è più efficiente che il 15% di BERT: le patch di un'immagine sono altamente ridondanti, richiedendo un masking più aggressivo per evitare la "copia" locale.

30.4 Fondamento Teorico: Augmentation Graph

Perché il contrastive learning impara features semantiche? L'intuizione formale (Arora et al., 2019): considera un grafo in cui i nodi sono le immagini e gli archi collegano le augmentazioni della stessa immagine. Il contrastive learning impara a mettere vicini i nodi connessi e lontani i disconnessi. Se le augmentazioni preservano la semantica, le rappresentazioni apprese catturano le features semantiche invarianti alle trasformazioni di augmentation.

PARTE 31

Meta-Learning: Imparare ad Imparare

31.1 Il Problema del Meta-Learning

L'obiettivo: imparare una policy di apprendimento (o un'inizializzazione) che permetta di adattarsi a nuovi task con pochi esempi. Un meta-dataset è una distribuzione di task $p(\mathcal{T})$; ogni task $\mathcal{T}_i = (\mathcal{D}_i^{\text{support}}, \mathcal{D}_i^{\text{query}})$ ha un piccolo support set per il fine-tuning e un query set per la valutazione.

31.2 MAML — Derivazione Completa

MAML (Finn et al., 2017) impara un'inizializzazione θ tale che, dopo pochi passi di gradient descent su un nuovo task, il modello funzioni bene. L'algoritmo:

MAML — INNER E OUTER LOOP

Inner loop (adattamento al task $\mathcal{T}_i$ con k passi GD):

$$\theta'_i = \theta - \alpha \nabla_{\theta} \mathcal{L}_{\mathcal{T}_i}^{\text{support}}(\theta)$$

Outer loop (meta-aggiornamento su tutti i task):

$$\theta \leftarrow \theta - \beta \nabla_{\theta} \sum_i \mathcal{L}_{\mathcal{T}_i}^{\text{query}}(\theta'_i)$$

Gradiente del secondo ordine

Il gradiente dell'outer loop rispetto a θ richiede di differenziare attraverso il passo GD dell'inner loop — un **gradiente del secondo ordine**. Espandendo

$$\theta'_i = \theta - \alpha \nabla_{\theta} \mathcal{L}^{\text{support}}(\theta):$$

1
$$\frac{\partial \theta'_i}{\partial \theta} = \mathbf{I} - \alpha \nabla_{\theta}^2 \mathcal{L}^{\text{support}}(\theta)$$

Derivata di $\theta - \alpha \nabla_{\theta} \mathcal{L}$: il termine $\alpha \nabla_{\theta}^2 \mathcal{L}$ è l'Hessiana

2
$$\nabla_{\theta} \mathcal{L}^{\text{query}}(\theta'_i) = \frac{\partial \mathcal{L}^{\text{query}}}{\partial \theta'_i} \cdot \frac{\partial \theta'_i}{\partial \theta} = \nabla_{\theta'_i} \mathcal{L}^{\text{query}} \cdot (\mathbf{I} - \alpha \mathbf{H}^{\text{support}})$$

Regola della catena; $\mathbf{H}^{\text{support}} = \nabla_{\theta}^2 \mathcal{L}^{\text{support}}$

$$\nabla_{\theta} \mathcal{L}^{\text{query}}(\theta'_i) = \nabla_{\theta'_i} \mathcal{L}^{\text{query}} - \alpha \nabla_{\theta'_i} \mathcal{L}^{\text{query}} \cdot \mathbf{H}^{\text{support}}(\theta)$$

Il secondo termine è il prodotto vettore-Hessiana, calcolabile con un secondo backpropagation senza materializzare l'Hessiana ($O(|\theta|)$ invece di $O(|\theta|^2)$). La variante **FOMAML** ignora questo termine (first-order approximation):

$$\nabla_{\theta} \approx \nabla_{\theta'_i} \mathcal{L}^{\text{query}}, \text{ sorprendentemente quasi altrettanto efficace.}$$

31.3 Reptile

Reptile (Nichol et al., 2018) è ancora più semplice: dopo k passi GD su un task, sposta θ verso i pesi adattati:

$$\theta \leftarrow \theta + \varepsilon(\theta'_i - \theta)$$

Equivale a MAML al primo ordine ma senza bisogno di query set separati — usa tutti gli esempi sia per inner che outer loop.

PARTE 32

Knowledge Distillation

32.1 Soft Targets e Temperature

La distillazione (Hinton et al., 2015) trasferisce la "conoscenza" di un modello grande (teacher T) a uno piccolo (student S). Il teacher produce distribuzioni morbide sui logit:

$$q_k^T = \frac{\exp(z_k^T / \tau)}{\sum_j \exp(z_j^T / \tau)}$$

Con temperatura $\tau > 1$ le probabilità diventano più uniformi, rivelando le relazioni apprese (es: un "2" è più simile a un "7" che a un "1"). La loss combinata dello student è:

$$\mathcal{L} = (1 - \alpha)\mathcal{L}_{\text{CE}}(\hat{y}_S, y) + \alpha\tau^2 D_{\text{KL}}(q^T \| q^S)$$

Il fattore τ^2 compensa la riduzione della magnitudine dei gradienti causata dalla temperatura: poiché $\partial q_k^T / \partial z_j \propto 1/\tau$, il gradiente rispetto ai logit è proporzionale a $1/\tau^2$ — moltiplicare per τ^2 riporta la magnitudine alla scala naturale.

32.2 Distillazione degli Strati Intermedi (FitNets)

FitNets (Romero et al., 2015) impone che i feature map intermedi dello student mimino quelli del teacher:

$$\mathcal{L}_{\text{hint}} = \frac{1}{2} \|\mathbf{h}_S \mathbf{W}_r - \mathbf{h}_T\|^2$$

dove $\mathbf{W}_r$ è una matrice di regressor che adatta le dimensioni dello student a quelle del teacher. Il training avviene in due fasi: prima minimizza $\mathcal{L}_{\text{hint}}$ (fase guided), poi ottimizza la loss finale combinata.

32.3 Relational Knowledge Distillation

Park et al. (2019) propongono di trasferire le *relazioni* tra esempi invece dei singoli embedding. La loss di distanza (RKD-D) preserva le distanze pairwise:

$$\mathcal{L}_{\text{RKD-D}} = \sum_{(i,j) \in \mathcal{X}^2} \ell_\delta \left(\frac{d_T(\mathbf{h}_i^T, \mathbf{h}_j^T)}{\mu_T} - \frac{d_S(\mathbf{h}_i^S, \mathbf{h}_j^S)}{\mu_S} \right)$$

dove $d(\cdot, \cdot)$ è la distanza euclidea, μ è la distanza media nel batch, e ℓ_δ è la Huber loss. Normalizzare per la distanza media rende la loss invariante alla scala dello spazio.

PARTE 33

Quantizzazione e Pruning

Tecniche per ridurre dimensione e costo computazionale dei modelli mantenendo le prestazioni.

33.1 Quantizzazione: Teoria

La quantizzazione mappa pesi float32 in interi a bassa precisione (INT8, INT4).

Per una quantizzazione uniforme con b bit:

$$w_q = \text{round}\left(\frac{w - w_{\min}}{w_{\max} - w_{\min}} \cdot (2^b - 1)\right)$$
$$w_{\text{dequant}} = w_q \cdot \frac{w_{\max} - w_{\min}}{2^b - 1} + w_{\min}$$

L'errore di quantizzazione per elemento è $\varepsilon = w - w_{\text{dequant}}$. La quantizzazione asimmetrica usa $\text{scale} = (w_{\max} - w_{\min}) / (2^b - 1)$ e $\text{zero_point} = -\text{round}(w_{\min} / \text{scale})$.

Straight-Through Estimator (STE)

Il problema del gradiente: $\partial(\text{round}(x)) / \partial x = 0$ quasi ovunque. Lo **STE** (Bengio et al., 2013) approssima:

$$\frac{\partial \mathcal{L}}{\partial w} \approx \frac{\partial \mathcal{L}}{\partial w_q} \cdot \mathbf{1}[w_{\min} \leq w \leq w_{\max}]$$

Il gradiente passa "dritto" attraverso l'arrotondamento (troncato fuori dal range). Questo permette il training di reti quantizzate (Quantization-Aware Training, QAT).

33.2 Magnitude Pruning

Il pruning rimuove i pesi con magnitudine più piccola. La sparsità target s significa mantenere solo $(1 - s) \cdot |\theta|$ pesi. Per la magnitude pruning non strutturata:

$$m_i = \mathbf{1}[|w_i| > \text{threshold}(s)], \quad \tilde{w}_i = m_i \cdot w_i$$

Il threshold è il $(100 \cdot s)$ -percentile dei $|w_i|$. Il **Lottery Ticket Hypothesis** (Frankle e Carlin, 2019) afferma che ogni rete densa contiene una "winning ticket" — una sottorete con la stessa inizializzazione originale — che può essere addestrata da sola ottenendo le stesse prestazioni.

L0 e L1 Pruning

Alternativa basata sulla regolarizzazione: aggiungere una penalità $\|w\|_0$ (conta pesi non-zero) incoraggia sparsità, ma $\|w\|_0$ è non differenziabile. L1 è il rilassamento convesso: $\|w\|_1 = \sum_i |w_i|$. La L1 porta molti pesi esattamente a zero durante l'ottimizzazione — produce sparsità in modo differenziabile.

Neural ODEs

Interpretare le ResNet come solutori di equazioni differenziali ordinarie, ottenendo modelli a profondità continua.

34.1 ResNet come Discretizzazione di un'ODE

Un blocco residuale è:

$$\mathbf{h}_{t+1} = \mathbf{h}_t + f(\mathbf{h}_t, \theta_t)$$

Questo è esattamente il metodo di Eulero per risolvere l'ODE:

$$\frac{d\mathbf{h}(t)}{dt} = f(\mathbf{h}(t), \theta(t)), \quad \mathbf{h}(t+1) \approx \mathbf{h}(t) + f(\mathbf{h}(t), \theta(t))$$

Una Neural ODE (Chen et al., 2018) parametrizza direttamente il campo vettoriale f_θ e usa un **solutore ODE adattivo** per calcolare l'output:

$$\mathbf{h}(T) = \mathbf{h}(0) + \int_0^T f(\mathbf{h}(t), t, \theta) dt = \text{ODESolve}(f_\theta, \mathbf{h}_0, t_0, T)$$

34.2 Il Metodo Aggiunto (Adjoint Method)

Come calcolare i gradienti rispetto a θ e $\mathbf{h}(0)$ senza memorizzare tutte le attivazioni intermedie (memory-efficient)? Si introduce la variabile aggiunta $\mathbf{a}(t) = \partial \mathcal{L} / \partial \mathbf{h}(t)$ e si dimostra che soddisfa un'ODE all'indietro:

- 1 Definiamo $\mathbf{a}(t) \triangleq \partial \mathcal{L} / \partial \mathbf{h}(t)$; vogliamo $d\mathbf{a}/dt$

Definizione

- 2 Per la regola della catena attraverso l'evoluzione temporale:

$$\frac{d\mathbf{a}(t)}{dt} = -\mathbf{a}(t)^T \frac{\partial f(\mathbf{h}(t), t, \theta)}{\partial \mathbf{h}(t)}$$

Differenziazione della condizione di stazionarietà del Lagrangiano

- 3 Gradiente rispetto ai parametri:

$$\frac{\partial \mathcal{L}}{\partial \theta} = - \int_{t_0}^T \mathbf{a}(t)^T \frac{\partial f(\mathbf{h}(t), t, \theta)}{\partial \theta} dt$$

Integrazione del prodotto gradiente-Jacobiano nel tempo

$$\frac{d\mathbf{a}(t)}{dt} = -\mathbf{a}(t)^T \frac{\partial f}{\partial \mathbf{h}}, \quad \frac{\partial \mathcal{L}}{\partial \theta} = - \int_{t_0}^T \mathbf{a}(t)^T \frac{\partial f}{\partial \theta} dt$$

Il backward pass risolve un'altra ODE all'indietro in $[T, t_0]$, senza memorizzare $\mathbf{h}(t)$ per ogni t — memoria $O(1)$ invece di $O(\text{depth})$.

PARTE 35

Transfer Learning e Fine-Tuning Efficiente

35.1 Feature Extraction vs Full Fine-Tuning

Con un modello pre-addestrato f_θ e un task downstream, due strategie:

Feature extraction: θ è congelato, si addestra solo un head lineare $\mathbf{W}_{\text{head}}$ sull'output di $f_\theta(\mathbf{x})$. Efficiente ma limitato: sfrutta solo le feature già estratte.

Full fine-tuning: si aggiornano tutti i parametri θ con LR piccolo. Potente ma costoso: numero di parametri aggiornabili = $|\theta|$.

35.2 LoRA — Low-Rank Adaptation

LoRA (Hu et al., 2022) approssima l'aggiornamento dei pesi come una matrice a basso rango, riducendo drasticamente i parametri addestrabili. Per una matrice di pesi $\mathbf{W}_0 \in \mathbb{R}^{d \times k}$:

LoRA — DECOMPOSIZIONE A BASSO RANGO

$$\mathbf{W} = \mathbf{W}_0 + \Delta\mathbf{W} = \mathbf{W}_0 + \mathbf{B}\mathbf{A}$$

dove $\mathbf{A} \in \mathbb{R}^{r \times k}$ e $\mathbf{B} \in \mathbb{R}^{d \times r}$ con $r \ll \min(d, k)$.

Inizializzazione: $\mathbf{A} \sim \mathcal{N}(0, \sigma^2)$, $\mathbf{B} = \mathbf{0}$ (partenza da $\Delta\mathbf{W} = 0$).

Durante il forward pass: $h = \mathbf{W}_0 x + \frac{\alpha}{r} \mathbf{B}\mathbf{A}x$

Il fattore di scala α/r normalizza la magnitudine dell'aggiornamento rispetto al rango. $\mathbf{W}_0$ è congelato; solo $\mathbf{A}$ e $\mathbf{B}$ vengono aggiornati. Parametri addestrabili: $r(d + k)$ vs dk originale. Per $r = 8$, $d = k = 4096$: risparmio di $4096^2 / (8 \times 2 \times 4096) = 256\times$.

Rank ottimale

La scelta di r è un trade-off: rank basso = meno parametri ma meno espressività.

In pratica $r \in \{4, 8, 16\}$ funziona bene per la maggior parte dei task NLP. Il **LoRA**

effective rank può essere stimato analizzando gli autovalori di $\Delta \mathbf{W}$: se la maggior parte della "varianza" è catturata da pochi autovalori, il rank basso è sufficiente.

35.3 Altre Tecniche PEFT

Prompt Tuning: aggiunge token addestrabili $\mathbf{P} \in \mathbb{R}^{l \times d}$ (soft prompts) in input, congelando il modello. Per modelli $\geq 10B$ parametri, raggiunge le prestazioni del full fine-tuning con solo $l \times d \ll |\theta|$ parametri.

Prefix Tuning: aggiunge prefix addestrabili alle Key e Value di ogni layer di attenzione, aumentando il contesto effettivo.

Adapter: inserisce strati bottleneck ($\mathbf{W}_{\text{down}} \in \mathbb{R}^{d \times r}$, $\mathbf{W}_{\text{up}} \in \mathbb{R}^{r \times d}$) dopo ogni sublayer del Transformer. La residual connection garantisce che, all'inizializzazione, l'adapter sia quasi identità.

PARTE 36

RLHF e Alignment dei Modelli Linguistici

36.1 Reward Model

RLHF (Reinforcement Learning from Human Feedback) ha tre fasi. Prima si addestra un **reward model** r_ϕ su preferenze umane. Dati due output (y_w, y_l) per lo stesso prompt x (dove $w = \text{winner}$, $l = \text{loser}$), la probabilità che y_w sia preferito è modellata con Bradley-Terry:

$$P(y_w \succ y_l \mid x) = \sigma(r_\phi(x, y_w) - r_\phi(x, y_l))$$

La loss di training è la binary cross-entropy sulle preferenze:

$$\mathcal{L}_{\text{RM}} = -\mathbb{E}_{(x, y_w, y_l) \sim \mathcal{D}} [\log \sigma(r_\phi(x, y_w) - r_\phi(x, y_l))]$$

36.2 PPO su LLM

Il modello linguistico π_θ viene ottimizzato tramite PPO per massimizzare il reward, con un termine KL che impedisce di allontanarsi troppo dalla policy di riferimento π_{ref} :

$$\mathcal{L}_{\text{RLHF}}(\theta) = \mathbb{E}_{x \sim \mathcal{D}, y \sim \pi_\theta(\cdot|x)} [r_\phi(x, y) - \beta D_{\text{KL}}(\pi_\theta(\cdot|x) \parallel \pi_{\text{ref}}(\cdot|x))]$$

Il termine KL penalizza distribuzioni molto diverse dalla policy originale SFT (Supervised Fine-Tuned), prevenendo il **reward hacking** (trovare comportamenti con reward alto ma non desiderabili). Espandendo:

$$\mathcal{D}_{\text{KL}}(\pi_\theta \parallel \pi_{\text{ref}}) = -\sum_{t=1}^T \sum_{v \in \mathcal{V}} \pi_\theta(v|x, y_{t-1}) \log \frac{\pi_\theta(v|x, y_{t-1})}{\pi_{\text{ref}}(v|x, y_{t-1})}$$

36.3 DPO — Direct Preference Optimization

DPO (Rafailov et al., 2023) mostra che l'ottimo di RLHF ha una forma chiusa, che può essere usata per eliminare il reward model e ottimizzare

direttamente la policy. Partendo dalla soluzione ottimale di RLHF:

1 La policy ottimale è: $\pi^*(y|x) = \frac{1}{Z(x)} \pi_{\text{ref}}(y|x) \exp\left(\frac{r(x,y)}{\beta}\right)$

Soluzione KKT del problema variazionale RLHF

2 Invertendo: $r(x, y) = \beta \log \frac{\pi^*(y|x)}{\pi_{\text{ref}}(y|x)} + \beta \log Z(x)$

Risolvendo per r

3 Sostituendo nella loss Bradley-Terry (la $Z(x)$ si cancella nella differenza):

$$\mathcal{L}_{\text{DPO}} = -\mathbb{E} \left[\log \sigma \left(\beta \log \frac{\pi_{\theta}(y_w|x)}{\pi_{\text{ref}}(y_w|x)} - \beta \log \frac{\pi_{\theta}(y_l|x)}{\pi_{\text{ref}}(y_l|x)} \right) \right]$$

Sostituzione in $P(y_w \succ y_l)$ e semplificazione

$$\mathcal{L}_{\text{DPO}}(\pi_{\theta}; \pi_{\text{ref}}) = -\mathbb{E}_{\mathcal{D}} \left[\log \sigma \left(\beta \log \frac{\pi_{\theta}(y_w|x)}{\pi_{\text{ref}}(y_w|x)} - \beta \log \frac{\pi_{\theta}(y_l|x)}{\pi_{\text{ref}}(y_l|x)} \right) \right]$$

DPO è più stabile di RLHF+PPO (niente reward model separato, niente campionamento online) e spesso raggiunge prestazioni comparabili.

PARTE 37

Interpretabilità e Explainability

37.1 Gradient-Based Saliency

Il metodo più semplice: la saliency map per un input $\mathbf{x}$ e classe c è il gradiente della logit rispetto all'input:

$$\mathbf{S}_c(\mathbf{x}) = \left| \frac{\partial f_c(\mathbf{x})}{\partial \mathbf{x}} \right|$$

Indica quanto ogni pixel (o token) contribuisce alla predizione. Problema: i gradienti sono rumorosi e instabili.

Integrated Gradients (Sundararajan et al., 2017) integra i gradienti lungo un percorso lineare da una baseline $\mathbf{x}'$ all'input $\mathbf{x}$:

$$\text{IG}_i(\mathbf{x}) = (x_i - x'_i) \int_0^1 \frac{\partial f(\mathbf{x}' + \alpha(\mathbf{x} - \mathbf{x}'))}{\partial x_i} d\alpha$$

Soddisfa la proprietà di completezza: $\sum_i \text{IG}_i(\mathbf{x}) = f(\mathbf{x}) - f(\mathbf{x}')$.

37.2 SHAP — Shapley Values

SHAP (Lundberg e Lee, 2017) usa i valori di Shapley dalla teoria dei giochi per attribuire la contribuzione di ogni feature. Il valore di Shapley della feature i è:

$$\phi_i(f, \mathbf{x}) = \sum_{S \subseteq \mathcal{F} \setminus \{i\}} \frac{|S|!(|\mathcal{F}| - |S| - 1)!}{|\mathcal{F}|!} [f(S \cup \{i\}) - f(S)]$$

dove $\mathcal{F}$ è l'insieme di tutte le feature e $f(S)$ è la predizione del modello con solo le feature in S (le altre vengono marginalizzate). Soddisfa tre assiomi: efficienza ($\sum_i \phi_i = f(\mathbf{x}) - \mathbb{E}[f]$), simmetria, e linearità.

Il calcolo esatto è $O(2^{|\mathcal{F}|})$. DeepSHAP approssima efficientemente per reti neurali sfruttando la linearità locale.

37.3 Probing Classifiers

Un probing classifier testa se una proprietà linguistica p (es. POS tag, profondità sintattica) è codificata nelle rappresentazioni $\mathbf{h}^{(\ell)}$ di un layer ℓ . Si addestra un classificatore semplice (lineare o MLP piccolo) $g_\psi : \mathbf{h}^{(\ell)} \rightarrow p$ — se funziona bene, le informazioni su p sono "presenti" in quel layer.

La **probing accuracy** misura la quantità di informazione, ma non controlla quanto sia facilmente accessibile. Il **Minimum Description Length (MDL) probing** misura invece il costo in bit per trasmettere le label usando la rappresentazione:

$$\text{MDL}(\ell, p) = H(p) - I(\mathbf{h}^{(\ell)}; p)$$

37.4 Mechanistic Interpretability

L'obiettivo è identificare i **circuiti** — sottografi del modello responsabili di comportamenti specifici. Il metodo di **activation patching** misura la contribuzione di ogni componente:

Per un comportamento B che differisce tra input $\mathbf{x}_c$ (corretto) e $\mathbf{x}_p$ (perturbato):

$$\Delta B_{\text{patch}}(c) = B(\text{patch}(\mathbf{x}_p, \mathbf{x}_c, c)) - B(\mathbf{x}_p)$$

Si sostituisce l'attivazione del componente c nella run su $\mathbf{x}_p$ con quella della run su $\mathbf{x}_c$. Componenti con ΔB_{patch} alto sono causalmente responsabili del comportamento B .

PARTE 38

Training Distribuito

38.1 Data Parallelism

Il mini-batch viene diviso tra M GPU, ognuna mantiene una copia completa del modello. Ogni GPU calcola il gradiente sul suo sotto-batch, poi si esegue un **AllReduce** per mediare i gradienti:

$$\mathbf{g}_{\text{sync}} = \frac{1}{M} \sum_{m=1}^M \mathbf{g}_m$$

L'AllReduce Ring-AllReduce (usato da PyTorch DDP) ha complessità $O(2(M-1)/M \cdot |\theta|)$ per la comunicazione — quasi indipendente da M per modelli grandi.

38.2 Model Parallelism

Per modelli troppo grandi per stare su una GPU. Due varianti:

Tensor Parallelism (Megatron-LM): divide le singole matrici di pesi tra GPU. Per una matrice $\mathbf{W} \in \mathbb{R}^{d \times k}$ su M GPU, ogni GPU ha $\mathbf{W}_{[:,m:m+k/M]}$. Richiede AllReduce dopo ogni moltiplicazione matrice — comunicazione frequente ma ben ottimizzata.

Pipeline Parallelism: ogni GPU ha un sottoinsieme di layer. Il training usa microbatch per mascherare la latenza di comunicazione:

$$\text{Efficienza pipeline} = \frac{m}{m + p - 1} \approx 1 - \frac{p}{m}$$

dove m è il numero di microbatch e p è il numero di stadi pipeline. Con $m \gg p$ l'overhead diventa trascurabile (GPipe schedule).

38.3 ZeRO — Zero Redundancy Optimizer

ZeRO (Rajbhandari et al., 2020) elimina la ridondanza della memoria nel data parallelism. Invece di replicare tutto su ogni GPU, partiziona:

Stage	Cosa si partiziona	Risparmio memoria
ZeRO-1	Optimizer states (m_t, v_t)	$4\times$ (per Adam: 8 byte/param $\rightarrow$ 2 byte/GPU con M GPU)
ZeRO-2	+ Gradienti	$8\times$
ZeRO-3	+ Parametri del modello	$M\times$ lineare

ZeRO-3 partiziona anche i pesi: ogni GPU ha $|\theta|/M$ parametri. Richiede AllGather prima di ogni forward pass e ReduceScatter dopo ogni backward — comunicazione aggiuntiva di $O(2|\theta|)$ per batch.

PARTE 39

Stabilità Numerica e Precisione Mista

39.1 Formati a Virgola Mobile

Formato	Bit totali	Esponente	Mantissa	Range	Precisione
float32 (FP32)	32	8	23	$\pm 3.4 \times 10^{38}$	~7 cifre decimali
float16 (FP16)	16	5	10	± 65504	~3 cifre decimali
bfloat16 (BF16)	16	8	7	$\pm 3.4 \times 10^{38}$	~2 cifre decimali
float8 E4M3	8	4	3	± 448	~1 cifra decimale

BF16 ha lo stesso range di FP32 (stesso esponente) ma meno precisione. FP16 ha più precisione ma range ridotto — overflow frequente con attivazioni grandi.

39.2 Mixed Precision Training

La strategia standard (Micikevicius et al., 2018):

- Pesi mantenuti in FP32 (master copy) e in FP16 (copia di computazione)
- Forward e backward in FP16 — $2\times$ throughput su GPU moderne
- Aggiornamenti dei pesi in FP32 — evita underflow dei piccoli gradienti

Il problema: i gradienti FP16 possono andare in **underflow** (troppo piccoli per essere rappresentati). La soluzione è il **loss scaling**: moltiplicare la loss per un fattore S prima del backward, poi dividere i gradienti per S :

$$\hat{\mathcal{L}} = S \cdot \mathcal{L}, \quad \nabla_{\theta} \mathcal{L} = \frac{1}{S} \nabla_{\theta} \hat{\mathcal{L}}$$

Il fattore S viene adattato dinamicamente: se si detectano overflow nei gradienti, S viene dimezzato; ogni N step senza overflow, S viene raddoppiato.

39.3 Trucchi Numerici

Log-sum-exp stabile: per evitare overflow nel calcolo $\log \sum_k e^{z_k}$:

$$\log \sum_k e^{z_k} = m + \log \sum_k e^{z_k - m}, \quad m = \max_k z_k$$

Softmax stabile: stessa idea — sottrarre il massimo prima di calcolare gli esponenti (non cambia il risultato perché il massimo si cancella nel

rapporto).

Calcolo stabile della varianza: la formula $\text{Var}(x) = \mathbb{E}[x^2] - (\mathbb{E}[x])^2$ è numericamente instabile per valori grandi. Si usa invece l'algoritmo di Welford:

$$\mu_n = \mu_{n-1} + \frac{x_n - \mu_{n-1}}{n}, \quad M_n = M_{n-1} + (x_n - \mu_{n-1})(x_n - \mu_n),$$
$$\sigma_n^2 = M_n / (n - 1)$$

Mantiene la varianza accumulata come $M_n = \sum_{i=1}^n (x_i - \mu_n)^2$ senza mai sottrarre numeri grandi.

PARTE 40

Continual Learning e Catastrophic Forgetting

40.1 Il Problema del Catastrophic Forgetting

Quando si addestra una rete su un task $\mathcal{T}_2$ dopo averla addestrata su $\mathcal{T}_1$, le prestazioni su $\mathcal{T}_1$ degradano — la rete "dimentica". Formalmente, se θ_1^* è l'ottimo su $\mathcal{T}_1$ e aggiornando verso θ_2^* si sposta lontano da θ_1^* :

$$\text{Forgetting}(\mathcal{T}_1) = \mathcal{L}_{\mathcal{T}_1}(\theta_2^*) - \mathcal{L}_{\mathcal{T}_1}(\theta_1^*) \geq 0$$

Il fenomeno è dovuto al fatto che le reti neurali usano rappresentazioni distribuite e condivise — aggiornare i pesi per un task modifica la computazione per tutti gli altri.

40.2 EWC — Elastic Weight Consolidation

EWC (Kirkpatrick et al., 2017) protegge i pesi importanti per il task precedente penalizzando i grandi spostamenti. L'importanza di ogni peso θ_i per il task $\mathcal{T}_1$ è stimata dalla diagonale della Fisher Information Matrix:

$$F_i = \mathbb{E}_{p(\mathbf{x}|\mathcal{T}_1)} \left[\left(\frac{\partial \log p_\theta(y|\mathbf{x})}{\partial \theta_i} \right)^2 \right] \approx \frac{1}{N} \sum_{n=1}^N \left(\frac{\partial \log p_\theta(y_n|\mathbf{x}_n)}{\partial \theta_i} \right)^2$$

La loss regolarizzata per il task $\mathcal{T}_2$ è:

$$\mathcal{L}_{\text{EWC}}(\theta) = \mathcal{L}_{\mathcal{T}_2}(\theta) + \frac{\lambda}{2} \sum_i F_i (\theta_i - \theta_{1,i}^*)^2$$

Il termine EWC è una approssimazione quadratica della KL divergence tra la posterior attuale e quella dopo il task 1:

$$\mathcal{L}_{\text{EWC}} \approx \mathcal{L}_{\mathcal{T}_2} + D_{\text{KL}}(p(\theta|\mathcal{T}_1) \| p(\theta)).$$

Connessione con la geometria della loss surface

La Fisher Information è uguale all'Hessiana della log-likelihood negativa in senso atteso: $F_i \approx -\mathbb{E}[\partial^2 \log p / \partial \theta_i^2]$. Quindi EWC approssima la curvatura della loss del task precedente — i pesi "importanti" sono quelli che la loss del task 1 penalizza maggiormente se spostati.

40.3 Experience Replay

Conservare un buffer $\mathcal{B}$ di esempi dai task precedenti e includerli nel training del task corrente:

$$\mathcal{L}_{\text{replay}}(\theta) = \mathcal{L}_{\mathcal{T}_{\text{new}}}(\theta) + \frac{|\mathcal{B}|}{|\mathcal{D}_{\text{new}}| + |\mathcal{B}|} \mathcal{L}_{\mathcal{B}}(\theta)$$

Generative Replay: invece di conservare dati reali, si addestra un modello generativo G_1 su $\mathcal{T}_1$ e si usa per generare "pseudo-esempi" durante il training su $\mathcal{T}_2$. Evita la memorizzazione di dati privati ma introduce il costo del modello generativo.

PARTE 41

Backpropagation: Derivazione Tensoriale Completa

Derivazione rigorosa dei gradienti per ogni operazione, dal caso scalare al batch matriciale, con notazione consistente.

41.1 Caso Scalare — Un Neurone

Partiamo dal caso più semplice: un singolo neurone con input $x \in \mathbb{R}$, peso $w \in \mathbb{R}$, bias $b \in \mathbb{R}$, attivazione sigmoid e loss MSE.

$$z = wx + b, \quad a = \sigma(z), \quad \mathcal{L} = \frac{1}{2}(a - y)^2$$

Calcoliamo ogni gradiente per la regola della catena, dall'output verso l'input:

$$\textcircled{1} \quad \frac{\partial \mathcal{L}}{\partial a} = a - y$$

Derivata di $\frac{1}{2}(a - y)^2$ rispetto ad a : $\frac{1}{2} \cdot 2(a - y) \cdot 1 = a - y$

$$\textcircled{2} \quad \frac{\partial a}{\partial z} = \sigma(z)(1 - \sigma(z)) = a(1 - a)$$

Derivata della sigmoid calcolata in Sezione 1.3

$$\textcircled{3} \quad \frac{\partial \mathcal{L}}{\partial z} = \frac{\partial \mathcal{L}}{\partial a} \cdot \frac{\partial a}{\partial z} = (a - y) \cdot a(1 - a)$$

Regola della catena: moltiplico i due gradienti

$$\textcircled{4} \quad \frac{\partial z}{\partial w} = x, \quad \frac{\partial z}{\partial b} = 1, \quad \frac{\partial z}{\partial x} = w$$

Derivate di $z = wx + b$ rispetto a ciascuna variabile

$$\textcircled{5} \quad \frac{\partial \mathcal{L}}{\partial w} = (a - y) \cdot a(1 - a) \cdot x$$

Catena completa: $\frac{\partial \mathcal{L}}{\partial z} \cdot \frac{\partial z}{\partial w}$

$$\textcircled{6} \quad \frac{\partial \mathcal{L}}{\partial b} = (a - y) \cdot a(1 - a), \quad \frac{\partial \mathcal{L}}{\partial x} = (a - y) \cdot a(1 - a) \cdot w$$

Analogo al passo 5

41.2 Caso Vettoriale — Un Layer

Layer con n_{in} input e n_{out} output, attivazione ReLU, loss MSE vettoriale.

Dati: $\mathbf{x} \in \mathbb{R}^{n_{\text{in}}}$, $\mathbf{W} \in \mathbb{R}^{n_{\text{out}} \times n_{\text{in}}}$, $\mathbf{b} \in \mathbb{R}^{n_{\text{out}}}$, target $\mathbf{y} \in \mathbb{R}^{n_{\text{out}}}$.

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}, \quad \mathbf{a} = \text{ReLU}(\mathbf{z}), \quad \mathcal{L} = \frac{1}{2} \|\mathbf{a} - \mathbf{y}\|^2$$

Gradiente rispetto a $\mathbf{a}$ (vettore n_{out})

$$\frac{\partial \mathcal{L}}{\partial a_i} = a_i - y_i \quad \Rightarrow \quad \frac{\partial \mathcal{L}}{\partial \mathbf{a}} = \mathbf{a} - \mathbf{y}$$

Gradiente rispetto a $\mathbf{z}$ — attraverso ReLU

ReLU è applicata componente per componente. Poiché ogni a_i dipende solo da z_i (non da z_j per $j \neq i$), la Jacobiana è diagonale:

$$\frac{\partial a_i}{\partial z_j} = \mathbf{1}[z_i > 0] \cdot \delta_{ij} \quad \Rightarrow \quad \frac{\partial \mathcal{L}}{\partial z_i} = \frac{\partial \mathcal{L}}{\partial a_i} \cdot \mathbf{1}[z_i > 0] = (a_i - y_i) \cdot \mathbf{1}$$

In notazione compatta con il prodotto di Hadamard $\odot$:

$$\boldsymbol{\delta} \triangleq \frac{\partial \mathcal{L}}{\partial \mathbf{z}} = (\mathbf{a} - \mathbf{y}) \odot \mathbf{1}[\mathbf{z} > 0]$$

Gradiente rispetto a $\mathbf{W}$ (matrice $n_{\text{out}} \times n_{\text{in}}$)

Ogni elemento W_{ij} contribuisce solo a $z_i = \sum_k W_{ik}x_k + b_i$:

$$\frac{\partial z_i}{\partial W_{ij}} = x_j \quad \Rightarrow \quad \frac{\partial \mathcal{L}}{\partial W_{ij}} = \frac{\partial \mathcal{L}}{\partial z_i} \cdot \frac{\partial z_i}{\partial W_{ij}} = \delta_i \cdot x_j$$

Scrivendo tutti gli elementi: la matrice $\partial\mathcal{L}/\partial\mathbf{W}$ ha elemento (i, j) uguale a $\delta_i x_j$ — questo è esattamente il prodotto esterno $\boldsymbol{\delta}\mathbf{x}^T$:

$$\frac{\partial\mathcal{L}}{\partial\mathbf{W}} = \boldsymbol{\delta}\mathbf{x}^T \in \mathbb{R}^{n_{\text{out}} \times n_{\text{in}}}$$

Gradiente rispetto a b

$$\frac{\partial z_i}{\partial b_i} = 1 \quad \Rightarrow \quad \frac{\partial\mathcal{L}}{\partial\mathbf{b}} = \boldsymbol{\delta}$$

Gradiente rispetto a x (per propagare al layer precedente)

$$\frac{\partial z_i}{\partial x_j} = W_{ij} \quad \Rightarrow \quad \frac{\partial\mathcal{L}}{\partial x_j} = \sum_i \frac{\partial\mathcal{L}}{\partial z_i} \cdot W_{ij} = \sum_i \delta_i W_{ij} = (\mathbf{W}^T \boldsymbol{\delta})_j$$

$$\frac{\partial\mathcal{L}}{\partial\mathbf{x}} = \mathbf{W}^T \boldsymbol{\delta}$$

41.3 Caso Batch — Mini-Batch con m Esempi

Con un mini-batch di m esempi, gli input sono colonne di $\mathbf{X} \in \mathbb{R}^{n_{\text{in}} \times m}$ e le pre-attivazioni sono $\mathbf{Z} = \mathbf{W}\mathbf{X} + \mathbf{b}\mathbf{1}^T \in \mathbb{R}^{n_{\text{out}} \times m}$. La loss è la media:

$$\mathcal{L} = \frac{1}{m} \sum_{k=1}^m \mathcal{L}^{(k)}$$

I gradienti diventano:

$$\begin{aligned}\Delta &= \frac{\partial \mathcal{L}}{\partial \mathbf{Z}} \in \mathbb{R}^{n_{\text{out}} \times m} \\ \frac{\partial \mathcal{L}}{\partial \mathbf{W}} &= \frac{1}{m} \Delta \mathbf{X}^T \in \mathbb{R}^{n_{\text{out}} \times n_{\text{in}}} \\ \frac{\partial \mathcal{L}}{\partial \mathbf{b}} &= \frac{1}{m} \Delta \mathbf{1} = \frac{1}{m} \sum_{k=1}^m \delta^{(k)} \in \mathbb{R}^{n_{\text{out}}} \\ \frac{\partial \mathcal{L}}{\partial \mathbf{X}} &= \mathbf{W}^T \Delta \in \mathbb{R}^{n_{\text{in}} \times m}\end{aligned}$$

Il gradiente di $\mathbf{b}$ è la *media* dei gradienti su tutti gli esempi — il broadcasting del bias viene "de-fatto" sommando lungo la dimensione batch.

PARTE 42

Adam: Analisi Matematica Profonda

Derivazione dell'aggiornamento ottimale, analisi della correzione del bias, convergenza per funzioni convesse, e confronto con Newton.

42.1 Motivazione: L'Aggiornamento Adattivo Ottimale

Consideriamo il problema di minimizzare $f(\theta)$ con informazione locale di primo ordine. Il passo di Newton usa l'Hessiana: $\theta_{t+1} = \theta_t - \mathbf{H}_t^{-1} \nabla f_t$. Per

funzioni quadratiche converte in un passo, ma richiede $O(d^2)$ memoria e $O(d^3)$ calcolo per invertire $\mathbf{H}$.

Adam approssima $\mathbf{H}^{-1}$ con una matrice diagonale $\hat{\mathbf{V}}_t^{-1/2}$, dove $\hat{V}_{t,i} = \hat{v}_{t,i}$ stima la varianza del gradiente nella direzione i . La giustificazione: per una funzione separabile $f(\theta) = \sum_i f_i(\theta_i)$, l'Hessiana è diagonale con $H_{ii} = f''_i$, e la varianza del gradiente $\mathbb{E}[g_i^2]$ approssima la curvatura locale f''_i (sotto condizioni di rumore specifiche).

42.2 Derivazione Rigida della Correzione del Bias

Il primo momento m_t è una media mobile esponenziale:

$m_t = \sum_{j=1}^t (1 - \beta_1) \beta_1^{t-j} g_j$. Calcoliamo il suo valore atteso, assumendo g_j stazionario con media $\mathbb{E}[g_j] = \bar{g}$:

$$1 \quad \mathbb{E}[m_t] = \sum_{j=1}^t (1 - \beta_1) \beta_1^{t-j} \mathbb{E}[g_j] = \bar{g} (1 - \beta_1) \sum_{j=1}^t \beta_1^{t-j}$$

Linearità della speranza; g_j stazionario

$$2 \quad \sum_{j=1}^t \beta_1^{t-j} = \beta_1^{t-1} + \beta_1^{t-2} + \dots + 1 = \frac{1 - \beta_1^t}{1 - \beta_1}$$

Somma geometrica: $\sum_{k=0}^{t-1} r^k = (1 - r^t)/(1 - r)$

$$3 \quad \mathbb{E}[m_t] = \bar{g} (1 - \beta_1) \cdot \frac{1 - \beta_1^t}{1 - \beta_1} = \bar{g} (1 - \beta_1^t)$$

Semplificazione

$$4 \quad \mathbb{E}[\hat{m}_t] = \mathbb{E}\left[\frac{m_t}{1 - \beta_1^t}\right] = \frac{\bar{g} (1 - \beta_1^t)}{1 - \beta_1^t} = \bar{g}$$

La correzione elimina esattamente il bias

Identicamente per v_t con β_2 e $\bar{g}^2 = \mathbb{E}[g_j^2]$. La correzione è *esatta* (non un'approssimazione) per processi stazionari.

42.3 Analisi del Regret — Convergenza Online

Nell'impostazione online, al passo t si riceve una loss convessa f_t . Il **regret** misura la differenza cumulativa rispetto al miglior punto fisso θ^* :

$$R_T = \sum_{t=1}^T f_t(\theta_t) - \sum_{t=1}^T f_t(\theta^*)$$

Per Adam (con AMSGrad per garantire la convergenza), il bound sul regret è:

$$R_T \leq \frac{\sqrt{T}}{1 - \beta_1} \sum_{i=1}^d \frac{\|\theta_{1:T,i} - \theta_i^*\|^2 \sqrt{\hat{v}_{T,i}}}{2\alpha} + \frac{\alpha}{(1 - \beta_1)\sqrt{1 - \beta_2}} \sum_{t=1}^T \sum_{i=1}^d$$

Il regret medio $R_T/T = O(1/\sqrt{T})$ garantisce convergenza al tasso $O(1/\sqrt{T})$ per il gap ottimale in media.

42.4 Perché Adam non Converge — e AMSGrad

Reddi et al. (2018) costruirono un controesempio per Adam su una sequenza di loss convesse. Il problema: il learning rate effettivo $\alpha_t^{(i)} = \alpha/\sqrt{\hat{v}_{t,i}}$ può *aumentare* nel tempo per alcune coordinate, anche quando i gradienti sono grandi. Con $\beta_2 = 0.999$, $\hat{v}_t$ decade lentamente — se il gradiente è grande solo occasionalmente, $\hat{v}_t$ si riduce tra gli eventi, aumentando il learning rate nelle fasi "calme". Questo può causare un comportamento oscillatorio che non converge.

La correzione AMSGrad sostituisce $\hat{v}_t$ con il suo massimo storico:

$$\hat{v}_{\max,t} = \max(\hat{v}_{\max,t-1}, \hat{v}_t) \geq \hat{v}_t$$

$$\theta_t = \theta_{t-1} - \frac{\alpha \hat{m}_t}{\sqrt{\hat{v}_{\max,t} + \varepsilon}}$$

Poiché $\hat{v}_{\max,t}$ è non decrescente, $\alpha_t^{(i)} = \alpha / \sqrt{\hat{v}_{\max,t,i}}$ è non crescente — il learning rate effettivo non può aumentare. Questo garantisce la convergenza anche nel caso online non-convesso.

PARTE 43

Meccanismo di Attenzione: Analisi

Profonda

Interpretazione del softmax come proiettore, analisi degli autovalori, Performer e kernel attention, e derivazione completa del backward pass.

43.1 La Softmax come Proiettore sul Simpleso

Il simpleso standard in $\mathbb{R}^n$ è $\Delta^{n-1} = \{\mathbf{p} \in \mathbb{R}^n : p_i \geq 0, \sum_i p_i = 1\}$. La softmax è una mappa differenziabile $\mathbb{R}^n \rightarrow \text{int}(\Delta^{n-1})$ (interno del simpleso). Proprietà importanti:

- **Temperature** $\rightarrow 0$: la softmax converge alla proiezione sul vertice più vicino (argmax) — distribuzione one-hot

- **Temperature** $\rightarrow \infty$: converge al centro del semplice $\mathbf{1}/n$ — distribuzione uniforme
- **Invarianza traslazionale**: $\text{softmax}(\mathbf{z} + c) = \text{softmax}(\mathbf{z})$ per qualsiasi costante c — usato per la stabilità numerica

43.2 Backward Pass dell'Attention —

Derivazione Completa

Dato $\mathbf{O} = \text{softmax}(\mathbf{S}/\sqrt{d_k})\mathbf{V}$ con $\mathbf{S} = \mathbf{Q}\mathbf{K}^T$, e $\partial\mathcal{L}/\partial\mathbf{O}$, calcoliamo i gradienti rispetto a $\mathbf{Q}, \mathbf{K}, \mathbf{V}$.

Definiamo $\mathbf{A} = \text{softmax}(\mathbf{S}/\sqrt{d_k}) \in \mathbb{R}^{n \times n}$ (matrice dei pesi di attenzione). Allora $\mathbf{O} = \mathbf{A}\mathbf{V}$.

Gradiente rispetto a $\mathbf{V}$

$$\frac{\partial\mathcal{L}}{\partial\mathbf{V}} = \mathbf{A}^T \frac{\partial\mathcal{L}}{\partial\mathbf{O}}$$

Diretta dalla regola $\partial(\mathbf{A}\mathbf{V})/\partial\mathbf{V} = \mathbf{A}^T$.

Gradiente rispetto a $\mathbf{A}$

$$\frac{\partial\mathcal{L}}{\partial\mathbf{A}} = \frac{\partial\mathcal{L}}{\partial\mathbf{O}} \mathbf{V}^T$$

Gradiente attraverso la Softmax per riga

Per la riga i di $\mathbf{A}$: $\mathbf{a}_i = \text{softmax}(\mathbf{s}_i/\sqrt{d_k})$. La Jacobiana della softmax è:

$$\frac{\partial a_{ij}}{\partial s_{ik}} = a_{ij}(\delta_{jk} - a_{ik})/\sqrt{d_k}$$

Il gradiente rispetto alla riga $\mathbf{s}_i$ è:

$$1 \quad \frac{\partial \mathcal{L}}{\partial s_{ik}} = \frac{1}{\sqrt{d_k}} \sum_j \frac{\partial \mathcal{L}}{\partial a_{ij}} \cdot a_{ij}(\delta_{jk} - a_{ik})$$

Regola della catena attraverso la Jacobiana

$$2 \quad = \frac{1}{\sqrt{d_k}} \left[a_{ik} \frac{\partial \mathcal{L}}{\partial a_{ik}} - a_{ik} \sum_j \frac{\partial \mathcal{L}}{\partial a_{ij}} a_{ij} \right]$$

Separando i termini $j = k$ e $j \neq k$

$$3 \quad \text{In forma vettoriale per la riga } i:$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{s}_i} = \frac{1}{\sqrt{d_k}} \mathbf{a}_i \odot \left(\frac{\partial \mathcal{L}}{\partial \mathbf{a}_i} - \left(\mathbf{a}_i^T \frac{\partial \mathcal{L}}{\partial \mathbf{a}_i} \right) \mathbf{1} \right)$$

Raccogliendo: il termine scalare $\mathbf{a}_i^T (\partial \mathcal{L} / \partial \mathbf{a}_i)$ è lo stesso per ogni k

$$\frac{\partial \mathcal{L}}{\partial \mathbf{S}} = \frac{1}{\sqrt{d_k}} \mathbf{A} \odot \left(\frac{\partial \mathcal{L}}{\partial \mathbf{A}} - \text{diag} \left(\frac{\partial \mathcal{L}}{\partial \mathbf{A}} \odot \mathbf{A} \cdot \mathbf{1} \right) \right)$$

Gradienti rispetto a $\mathbf{Q}$ e $\mathbf{K}$

Da $\mathbf{S} = \mathbf{QK}^T$:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{Q}} = \frac{\partial \mathcal{L}}{\partial \mathbf{S}} \mathbf{K}, \quad \frac{\partial \mathcal{L}}{\partial \mathbf{K}} = \left(\frac{\partial \mathcal{L}}{\partial \mathbf{S}} \right)^T \mathbf{Q}$$

43.3 Performer: Kernel Attention Lineare

L'attenzione standard ha complessità $O(n^2d)$. Per sequenze lunghe occorre linearizzarla. Il Performer (Choromanski et al., 2021) approssima il kernel softmax con features random.

Osservazione chiave: il kernel softmax $K(\mathbf{q}, \mathbf{k}) = \exp(\mathbf{q}^T \mathbf{k} / \sqrt{d})$ può essere decomposto come:

$$K(\mathbf{q}, \mathbf{k}) = \mathbb{E}_{\boldsymbol{\omega} \sim \mathcal{N}(0, \mathbf{I}_d)} \left[\exp\left(\boldsymbol{\omega}^T \mathbf{q} / \sqrt[4]{d} - \|\mathbf{q}\|^2 / (2\sqrt{d})\right) \cdot \exp\left(\boldsymbol{\omega}^T \mathbf{k} / \sqrt[4]{d}\right) \right]$$

Con m features random $\boldsymbol{\omega}_1, \dots, \boldsymbol{\omega}_m$, si definisce la feature map:

$$\phi(\mathbf{q}) = \frac{1}{\sqrt{m}} \exp\left(-\frac{\|\mathbf{q}\|^2}{2\sqrt{d}}\right) \left[\exp\left(\frac{\boldsymbol{\omega}_1^T \mathbf{q}}{\sqrt[4]{d}}\right), \dots, \exp\left(\frac{\boldsymbol{\omega}_m^T \mathbf{q}}{\sqrt[4]{d}}\right) \right]^T \in \mathbb{R}^m$$

Allora $K(\mathbf{q}, \mathbf{k}) \approx \phi(\mathbf{q})^T \phi(\mathbf{k})$, e l'attenzione diventa:

$$\mathbf{O}_i \approx \frac{\phi(\mathbf{q}_i)^T \left(\sum_j \phi(\mathbf{k}_j) \mathbf{v}_j^T \right)}{\phi(\mathbf{q}_i)^T \left(\sum_j \phi(\mathbf{k}_j) \right)}$$

Le somme $\sum_j \phi(\mathbf{k}_j) \mathbf{v}_j^T \in \mathbb{R}^{m \times d}$ e $\sum_j \phi(\mathbf{k}_j) \in \mathbb{R}^m$ si calcolano una volta sola: complessità totale $O(nmd)$ invece di $O(n^2d)$.

LSTM: Analisi Dinamica e Backpropagation Completa

44.1 Analisi dello Stato della Cella come Sistema Dinamico

L'equazione di update della cella LSTM è $\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t$.

Possiamo analizzarla come un sistema dinamico lineare a tempo variante in ogni dimensione j :

$$c_{t,j} = f_{t,j} \cdot c_{t-1,j} + i_{t,j} \cdot \tilde{c}_{t,j}$$

L'autovalore locale al passo t è $f_{t,j} \in (0, 1)$. La soluzione esatta per la dimensione j :

$$c_{T,j} = c_{0,j} \prod_{t=1}^T f_{t,j} + \sum_{t=1}^T i_{t,j} \tilde{c}_{t,j} \prod_{s=t+1}^T f_{s,j}$$

Se $f_{t,j} \approx 1$: $c_{T,j} \approx c_{0,j} + \sum_t i_{t,j} \tilde{c}_{t,j}$ — la cella accumula informazione additivamente (memory a lungo termine). Se $f_{t,j} \approx 0$: la cella viene resettata — memory a breve termine. Il modello impara a modulare $f_{t,j}$ in base al contesto.

44.2 BPTT dell'LSTM — Derivazione Completa

Il backward pass dell'LSTM a un passo. Definiamo $\delta_h^{(t)} = \partial \mathcal{L} / \partial \mathbf{h}_t$ e $\delta_c^{(t)} = \partial \mathcal{L} / \partial \mathbf{c}_t$ (gradienti che arrivano dal passo successivo e dall'output).

Step 1 — Gradiente rispetto a $\mathbf{c}_t$

$$\delta_c^{(t)} = \delta_h^{(t)} \odot \mathbf{o}_t \odot (1 - \tanh^2(\mathbf{c}_t)) + \delta_c^{(t+1)} \odot \mathbf{f}_{t+1}$$

Il primo termine: gradiente da $\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$ con derivata $\mathbf{o}_t \odot \text{sech}^2(\mathbf{c}_t)$. Il secondo termine: gradiente che scorre dalla cella futura attraverso il forget gate.

Step 2 — Gradienti rispetto ai gate

$$\delta_{\tilde{\mathbf{c}}}^{(t)} = \delta_c^{(t)} \odot \mathbf{i}_t \odot (1 - \tilde{\mathbf{c}}_t^2) \quad (\text{tanh candidate})$$

$$\delta_i^{(t)} = \delta_c^{(t)} \odot \tilde{\mathbf{c}}_t \odot \mathbf{i}_t \odot (1 - \mathbf{i}_t) \quad (\text{input gate})$$

$$\delta_f^{(t)} = \delta_c^{(t)} \odot \mathbf{c}_{t-1} \odot \mathbf{f}_t \odot (1 - \mathbf{f}_t) \quad (\text{forget gate})$$

$$\delta_o^{(t)} = \delta_h^{(t)} \odot \tanh(\mathbf{c}_t) \odot \mathbf{o}_t \odot (1 - \mathbf{o}_t) \quad (\text{output gate})$$

Step 3 — Gradienti rispetto alle matrici dei pesi

Concatenando i gradienti dei quattro gate: $\Delta = [\delta_{\tilde{\mathbf{c}}}^{(t)}; \delta_i^{(t)}; \delta_f^{(t)}; \delta_o^{(t)}] \in \mathbb{R}^{4d}$.

Con input concatenato $\mathbf{u} = [\mathbf{h}_{t-1}; \mathbf{x}_t] \in \mathbb{R}^{d+n}$:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \Delta \mathbf{u}^T \in \mathbb{R}^{4d \times (d+n)}, \quad \frac{\partial \mathcal{L}}{\partial \mathbf{b}} = \Delta \in \mathbb{R}^{4d}$$

Step 4 — Gradiente rispetto a $\mathbf{h}_{t-1}$ (per propagare all'indietro)

$$\delta_h^{(t-1)} = \mathbf{W}_{hh}^T \Delta$$

dove $\mathbf{W}_{hh}$ sono le quattro sotto-matrici che moltiplicano $\mathbf{h}_{t-1}$.

PARTE 45

Softmax e Cross-Entropy: Proprietà Matematiche Profonde

45.1 Geometria della Softmax

La softmax può essere interpretata come la soluzione al problema di ottimizzazione vincolato: trovare la distribuzione $\mathbf{p} \in \Delta^{K-1}$ che massimizza l'entropia soggetta a un vincolo di energia media:

TEOREMA — SOFTMAX COME MAX-ENTROPIA

La soluzione di $\max_{\mathbf{p}} H(\mathbf{p}) = -\sum_k p_k \log p_k$ soggetta a $\sum_k p_k = 1$ e $\sum_k p_k z_k = c$ (costante) è:

$$p_k^* = \frac{e^{\lambda z_k}}{\sum_j e^{\lambda z_j}} = \text{softmax}(\lambda \mathbf{z})_k$$

dove λ è il moltiplicatore di Lagrange associato al vincolo sull'energia media.

Derivazione tramite Lagrangiano:

$$\mathcal{L}(\mathbf{p}, \lambda, \mu) = - \sum_k p_k \log p_k - \lambda \left(\sum_k p_k z_k - c \right) - \mu \left(\sum_k p_k - 1 \right)$$

$$\frac{\partial \mathcal{L}}{\partial p_k} = -\log p_k - 1 - \lambda z_k - \mu = 0 \implies p_k = e^{-1-\mu-\lambda z_k} = \frac{e^{\lambda z_k}}{e^{1+\mu}}$$

45.2 Cross-Entropy come Upper Bound sulla Log-Loss

Mostriamo che la cross-entropy è un upper bound stretto sulla log-loss vera $-\log p_{\text{data}}(\mathbf{x})$:

$$H(p_{\text{data}}, q_{\theta}) = H(p_{\text{data}}) + D_{\text{KL}}(p_{\text{data}} \| q_{\theta}) \geq H(p_{\text{data}})$$

Minimizzare la cross-entropy riduce $D_{\text{KL}}(p_{\text{data}} \| q_{\theta})$ verso zero — il modello converge verso la distribuzione dei dati. Quando $q_{\theta} = p_{\text{data}}$, la cross-entropy raggiunge il suo minimo $H(p_{\text{data}})$ (l'entropia vera dei dati).

45.3 Focal Loss — Generalizzazione della Cross-Entropy

Lin et al. (2017) propongono la Focal Loss per gestire il class imbalance. La cross-entropy binaria standard dà peso uguale a esempi facili e difficili. La focal loss riduce il contributo degli esempi facili (alta confidenza):

$$\mathcal{L}_{\text{focal}} = -(1 - p_t)^\gamma \log(p_t)$$

dove $p_t = \hat{y}$ se $y = 1$, $1 - \hat{y}$ se $y = 0$, e $\gamma \geq 0$ è il parametro di focusing.

Per $\gamma = 0$: ridiventa BCE. Per $\gamma = 2$ (default): esempi con $p_t = 0.9$

ricevono un peso $(0.1)^2 = 0.01 \times$ rispetto agli esempi difficili con $p_t = 0.1$, peso $(0.9)^2 \approx 0.81 \times$. Il gradiente:

$$\frac{\partial \mathcal{L}_{\text{focal}}}{\partial z} = (1 - p_t)^\gamma (\gamma p_t \log(p_t) - 1 + p_t) \cdot (p_t - y)$$

PARTE 46

Gradiente Naturale e Fisher Information

46.1 La Metrica di Fisher-Rao

Lo spazio dei parametri θ è uno spazio euclideo, ma lo spazio delle distribuzioni p_θ ha una geometria naturale diversa. La **Fisher Information Matrix** definisce una metrica Riemanniana su questo spazio:

$$\mathbf{F}(\theta) = \mathbb{E}_{p_\theta(\mathbf{x})} [\nabla_\theta \log p_\theta(\mathbf{x}) (\nabla_\theta \log p_\theta(\mathbf{x}))^T]$$

Due proprietà fondamentali: (1) $\mathbf{F}(\theta) \succeq 0$ (semidefinita positiva), (2)

$\mathbf{F}(\theta) = -\mathbb{E}[\nabla_\theta^2 \log p_\theta(\mathbf{x})]$ — la Fisher è uguale all'Hessiana negativa della log-likelihood in valore atteso.

Legame con la KL divergence

La Fisher è la curvatura locale della KL divergence:

$$D_{\text{KL}}(p_{\theta} \| p_{\theta+\delta}) = \frac{1}{2} \delta^T \mathbf{F}(\theta) \delta + O(\|\delta\|^3)$$

Quindi un passo nel spazio dei parametri $\delta\theta$ con $\|\delta\theta\|_{\mathbf{F}} = c$ corrisponde a un cambiamento costante c nella distribuzione (in KL) — indipendentemente dalla parametrizzazione.

46.2 L'Aggiornamento del Gradiente Naturale

Il gradiente ordinario minimizza la loss lungo la direzione di massima discesa nello spazio euclideo. Il **gradiente naturale** minimizza lungo la direzione di massima discesa nello spazio delle distribuzioni:

$$\begin{aligned}\tilde{\nabla}_{\theta} \mathcal{L} &= \mathbf{F}(\theta)^{-1} \nabla_{\theta} \mathcal{L} \\ \theta_{t+1} &= \theta_t - \alpha \mathbf{F}(\theta_t)^{-1} \nabla_{\theta} \mathcal{L}\end{aligned}$$

Invarianza alla reparametrizzazione

Questa è la proprietà chiave. Se si usa una parametrizzazione alternativa $\phi = g(\theta)$, l'aggiornamento del gradiente ordinario cambia; il gradiente naturale no. Formalmente, se $\tilde{\nabla}_{\theta} \mathcal{L}$ è il gradiente naturale in θ , allora:

$$\tilde{\nabla}_{\phi} \mathcal{L} = \mathbf{J}_g^{-1} \tilde{\nabla}_{\theta} \mathcal{L} \cdot \text{sgn}(\det \mathbf{J}_g)$$

Lo stesso passo nello spazio delle distribuzioni, indipendentemente dalla parametrizzazione. Questo è perché TRPO e PPO (che limitano i cambiamenti in D_{KL}) sono più stabili del gradiente ordinario.

46.3 K-FAC: Approssimazione Kronecker della Fisher

Per un layer FC con pesi $\mathbf{W} \in \mathbb{R}^{d_{out} \times d_{in}}$, il gradiente è $\mathbf{g} = \boldsymbol{\delta} \mathbf{a}^T$ (prodotto esterno di gradiente pre-attivazione e attivazione). La Fisher per i pesi di questo layer è:

$$\mathbf{F}_W = \mathbb{E}[\mathbf{g} \otimes \mathbf{g}] = \mathbb{E}[(\boldsymbol{\delta} \mathbf{a}^T) \otimes (\boldsymbol{\delta} \mathbf{a}^T)] = \mathbb{E}[\boldsymbol{\delta} \boldsymbol{\delta}^T] \otimes \mathbb{E}[\mathbf{a} \mathbf{a}^T]$$

L'ultima uguaglianza assume indipendenza tra $\boldsymbol{\delta}$ e $\mathbf{a}$ (approssimazione). Definendo:

$$\mathbf{A} = \mathbb{E}[\mathbf{a} \mathbf{a}^T] \in \mathbb{R}^{d_{in} \times d_{in}}, \quad \mathbf{G} = \mathbb{E}[\boldsymbol{\delta} \boldsymbol{\delta}^T] \in \mathbb{R}^{d_{out} \times d_{out}}$$

$$\mathbf{F}_W \approx \mathbf{G} \otimes \mathbf{A}$$

L'inversione sfrutta la proprietà del prodotto di Kronecker:

$$(\mathbf{G} \otimes \mathbf{A})^{-1} = \mathbf{G}^{-1} \otimes \mathbf{A}^{-1}$$

Costo: inversione di due matrici $d_{out} \times d_{out}$ e $d_{in} \times d_{in}$ invece di $d_{out} d_{in} \times d_{out} d_{in}$ — risparmio $O(d^2)$ per layer.

Normalizing Flows: Analisi Matematica Profonda

47.1 Derivazione Rigorosa del Cambio di Variabile

Dimostriamo la formula del cambio di variabile. Sia $f : \mathbb{R}^d \rightarrow \mathbb{R}^d$ differenziabile e invertibile, $\mathbf{z} \sim p_Z$, $\mathbf{x} = f(\mathbf{z})$. Per un insieme misurabile A :

- 1 $P(\mathbf{x} \in A) = P(\mathbf{z} \in f^{-1}(A)) = \int_{f^{-1}(A)} p_Z(\mathbf{z}) d\mathbf{z}$

Definizione: $\mathbf{x} \in A$ sse $\mathbf{z} = f^{-1}(\mathbf{x}) \in f^{-1}(A)$

- 2 Cambio di variabile $\mathbf{z} = f^{-1}(\mathbf{x})$, $d\mathbf{z} = |\det \mathbf{J}_{f^{-1}}(\mathbf{x})| d\mathbf{x}$:
 $= \int_A p_Z(f^{-1}(\mathbf{x})) |\det \mathbf{J}_{f^{-1}}(\mathbf{x})| d\mathbf{x}$

Teorema del cambio di variabile nell'integrale

- 3 Poiché $P(\mathbf{x} \in A) = \int_A p_X(\mathbf{x}) d\mathbf{x}$ per ogni A :

$$p_X(\mathbf{x}) = p_Z(f^{-1}(\mathbf{x})) \cdot |\det \mathbf{J}_{f^{-1}}(\mathbf{x})|$$

Identificazione delle densità

- 4 Usando $\mathbf{J}_{f^{-1}} = (\mathbf{J}_f)^{-1}$ e $\det(\mathbf{A}^{-1}) = 1/\det(\mathbf{A})$:

$$p_X(\mathbf{x}) = p_Z(f^{-1}(\mathbf{x})) \cdot |\det \mathbf{J}_f(f^{-1}(\mathbf{x}))|^{-1}$$

Proprietà della matrice inversa

$$\log p_X(\mathbf{x}) = \log p_Z(\mathbf{z}) - \log |\det \mathbf{J}_f(\mathbf{z})|, \quad \text{con } \mathbf{z} = f^{-1}(\mathbf{x})$$

47.2 Flow Autoregressivi

Il Masked Autoregressive Flow (MAF, Papamakarios et al., 2017) usa una trasformazione autoregressiva: ogni dimensione x_i dipende da quelle precedenti $x_{1:i-1}$:

$$x_i = z_i \cdot \exp(\alpha_i(x_{1:i-1})) + \mu_i(x_{1:i-1})$$

Il Jacobiano di questa trasformazione è **triangolare inferiore** — il suo determinante è il prodotto degli elementi diagonali:

$$\mathbf{J}_f = \begin{pmatrix} \exp(\alpha_1) & 0 & \cdots & 0 \\ \frac{\partial x_2}{\partial z_1} & \exp(\alpha_2) & \cdots & 0 \\ \vdots & & \ddots & 0 \\ \frac{\partial x_d}{\partial z_1} & \cdots & \cdots & \exp(\alpha_d) \end{pmatrix}$$

$$\log |\det \mathbf{J}_f| = \sum_{i=1}^d \alpha_i(x_{1:i-1})$$

Il forward pass (generazione) richiede d passi sequenziali. Il backward pass (calcolo della densità) è parallelizzabile grazie alla struttura autoregressiva.

48.1 Mean-Field VI — Coordinate Ascent

Per la famiglia mean-field $q(\theta) = \prod_i q_i(\theta_i)$ (fattorizzazione completa), l'ELBO si massimizza con l'algoritmo **Coordinate Ascent VI (CAVI)**: si ottimizza q_j tenendo fissi tutti gli altri q_i .

La distribuzione ottimale per il fattore j è:

TEOREMA CAVI

L'aggiornamento ottimale per $q_j(\theta_j)$ è:

$$\log q_j^*(\theta_j) = \mathbb{E}_{q_{-j}}[\log p(\theta, \mathbf{x})] + C_j$$

dove $\mathbb{E}_{q_{-j}}[\cdot]$ è il valore atteso rispetto a tutti i fattori tranne j , e C_j è la costante di normalizzazione.

Derivazione: L'ELBO è $\mathcal{L}(q) = \mathbb{E}_q[\log p(\theta, \mathbf{x})] - \mathbb{E}_q[\log q(\theta)]$. Isolando i termini che dipendono da q_j :

$$\begin{aligned}\mathcal{L}(q_j) &= \int q_j(\theta_j) \mathbb{E}_{q_{-j}}[\log p(\theta, \mathbf{x})] d\theta_j - \int q_j(\theta_j) \log q_j(\theta_j) d\theta_j + C \\ &= -D_{\text{KL}}(q_j \| \exp(\mathbb{E}_{q_{-j}}[\log p(\theta, \mathbf{x})])) + C'\end{aligned}$$

La KL è minimizzata (ELBO massimizzato) quando

$$q_j^* \propto \exp(\mathbb{E}_{q_{-j}}[\log p(\theta, \mathbf{x})]).$$

48.2 Importance Weighted Autoencoder (IWAE)

L'ELBO standard è un lower bound su $\log p(\mathbf{x})$. IWAE (Burda et al., 2016) fornisce un bound più stretto usando K campioni:

$$\mathcal{L}_K = \mathbb{E}_{\mathbf{z}^{1:K} \sim q_\phi} \left[\log \frac{1}{K} \sum_{k=1}^K \frac{p_\theta(\mathbf{x}, \mathbf{z}^k)}{q_\phi(\mathbf{z}^k | \mathbf{x})} \right]$$

Risultato: $\mathcal{L}_1 \leq \mathcal{L}_2 \leq \dots \leq \mathcal{L}_K \leq \log p(\mathbf{x})$. La sequenza converge a $\log p(\mathbf{x})$ per $K \rightarrow \infty$ (per la legge dei grandi numeri applicata all'estimatore di Monte Carlo).

Il gradiente dell'IWAE rispetto a ϕ usa i pesi importanza normalizzati

$\hat{w}_k = w_k / \sum_j w_j$ con $w_k = p_\theta(\mathbf{x}, \mathbf{z}^k) / q_\phi(\mathbf{z}^k | \mathbf{x})$:

$$\nabla_\phi \mathcal{L}_K = \mathbb{E} \left[\sum_k \hat{w}_k \nabla_\phi \log q_\phi(\mathbf{z}^k | \mathbf{x}) \right]$$

PARTE 49

Teoria della Convergenza: SGD, Rumore e Flat Minima

49.1 Analisi del Rumore in SGD

Il gradiente stocastico $\hat{\mathbf{g}} = \frac{1}{B} \sum_{i \in \mathcal{B}} \nabla_{\theta} \mathcal{L}^{(i)}$ è una stima del gradiente vero $\mathbf{g} = \nabla_{\theta} \mathcal{L}$. Il **noise del gradiente** è:

$$\boldsymbol{\eta} = \hat{\mathbf{g}} - \mathbf{g}, \quad \mathbb{E}[\boldsymbol{\eta}] = \mathbf{0}, \quad \text{Cov}(\boldsymbol{\eta}) = \frac{1}{B} \boldsymbol{\Sigma}(\theta)$$

dove $\boldsymbol{\Sigma}(\theta) = \text{Cov}_{i \sim \text{data}}(\nabla_{\theta} \mathcal{L}^{(i)}) = \mathbb{E}_i[(\nabla_{\theta} \mathcal{L}^{(i)})(\nabla_{\theta} \mathcal{L}^{(i)})^T] - \mathbf{g}\mathbf{g}^T$. Il rumore è inversamente proporzionale al batch size B .

Equazione di Fokker-Planck per SGD

Nel limite di learning rate piccolo, SGD è approssimato da una SDE (Stochastic Differential Equation):

$$d\theta = -\nabla \mathcal{L}(\theta) dt + \sqrt{\frac{\alpha}{B}} \boldsymbol{\Sigma}^{1/2}(\theta) d\mathbf{W}_t$$

dove $d\mathbf{W}_t$ è il moto Browniano. Questa SDE ha una distribuzione stazionaria proporzionale a

$p^*(\theta) \propto \exp(-2\mathcal{L}(\theta)/(\alpha/B)) = \exp(-2B\mathcal{L}(\theta)/\alpha)$ — una distribuzione di Gibbs con "temperatura" $T_{\text{eff}} = \alpha/(2B)$. Le implicazioni:

- Temperatura più alta (α grande, B piccolo) $\rightarrow$ maggiore esplorazione $\rightarrow$ tende a flat minima
- Temperatura più bassa $\rightarrow$ si concentra nei minimi più profondi

49.2 Sharpness e PAC-Bayes

Il framework PAC-Bayes (McAllester, 1999) fornisce un bound sull'errore di generalizzazione che coinvolge esplicitamente la sharpness. Per ogni distribuzione Q sui pesi:

$$\mathcal{L}_{\text{test}}(Q) \leq \mathcal{L}_{\text{train}}(Q) + \sqrt{\frac{D_{\text{KL}}(Q\|P) + \ln(N/\delta)}{2(N-1)}}$$

dove P è il prior e N è il numero di esempi. Scegliendo $Q = \mathcal{N}(\theta^*, \sigma^2 \mathbf{I})$ (distribuzione centrata sul minimo trovato con varianza σ^2):

$$D_{\text{KL}}(Q\|P) \propto \frac{\|\theta^*\|^2}{\sigma^2} + d \ln \frac{\sigma_0^2}{\sigma^2}$$

Per σ grande (flat minimum): D_{KL} è piccolo se l'energia

$\mathcal{L}_{\text{train}}(Q) \approx \mathcal{L}_{\text{train}}(\theta^*)$ — un flat minimum permette di scegliere σ grande mantenendo la loss bassa, migliorando il bound di generalizzazione.

49.3 Edge of Stability

Cohen et al. (2021) osservarono che con SGD a LR fisso, durante il training la sharpness (massimo autovalore dell'Hessiana $\lambda_{\text{max}}(\mathbf{H})$) cresce fino a $2/\alpha$ e poi si stabilizza — questo è l'"edge of stability".

La spiegazione: il passo di GD converge strettamente se $\lambda_{\text{max}} \leq 2/\alpha$ (condizione di Cauchy). Quando $\lambda_{\text{max}} > 2/\alpha$, l'aggiornamento diventa instabile in quella direzione, ma il rumore del mini-batch sposta il parametro in regioni con curvatura più bassa — il processo auto-regola la sharpness.

Diffusion Models: Derivazione Completa dell'ELBO

50.1 Proprietà del Processo Forward

Ricordiamo le definizioni: $q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1 - \bar{\alpha}_t)\mathbf{I})$ con $\bar{\alpha}_t = \prod_{s=1}^t \alpha_s$ e $\alpha_s = 1 - \beta_s$.

Proprietà fondamentale: la distribuzione della posterior $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ è analitica:

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1}; \tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0), \tilde{\beta}_t\mathbf{I})$$

dove:

$$\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0) = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1 - \bar{\alpha}_t}\mathbf{x}_0 + \frac{\sqrt{\bar{\alpha}_t}(1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t}\mathbf{x}_t, \quad \tilde{\beta}_t = \frac{(1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t}$$

Derivazione tramite Bayes

1 $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) \propto q(\mathbf{x}_t|\mathbf{x}_{t-1}) \cdot q(\mathbf{x}_{t-1}|\mathbf{x}_0)$

Teorema di Bayes + proprietà Markov

2 Entrambi i termini sono gaussiani. Il prodotto di gaussiane è gaussiano. Completando il quadrato nell'esponente si ottengono

$\tilde{\mu}_t$ e $\tilde{\beta}_t$.

Proprietà del prodotto di gaussiane

50.2 ELBO Completo del Modello di Diffusione

La log-likelihood è lower bounded dall'ELBO. Scriviamo la decomposizione VLB (Variational Lower Bound) per i diffusion model:

$$\log p_\theta(\mathbf{x}_0) \geq \mathbb{E}_q \left[\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \right] = -L_{\text{VLB}}$$

Espandendo con la struttura Markov:

$$L_{\text{VLB}} = \underbrace{D_{\text{KL}}(q(\mathbf{x}_T|\mathbf{x}_0)||p(\mathbf{x}_T))}_{L_T \approx 0} + \sum_{t=2}^T \underbrace{D_{\text{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)||p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))}_{L_{t-1}}$$

Semplificazione di L_{t-1}

Poiché entrambe le distribuzioni sono gaussiane, la KL è analitica. Il modello parametrizza $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \sigma_t^2 \mathbf{I})$ e si dimostra che:

$$L_{t-1} = \mathbb{E}_q \left[\frac{1}{2\sigma_t^2} \|\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0) - \boldsymbol{\mu}_\theta(\mathbf{x}_t, t)\|^2 \right] + C$$

Reparametrizzando $\boldsymbol{\mu}_\theta$ in termini del rumore predetto ϵ_θ :

$$\boldsymbol{\mu}_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right)$$

e $\tilde{\boldsymbol{\mu}}_t = \frac{1}{\sqrt{\alpha_t}} (\mathbf{x}_t - \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t}} \boldsymbol{\epsilon})$ (con $\boldsymbol{\epsilon}$ il rumore vero), allora:

$$L_{t-1} = \mathbb{E}_{\mathbf{x}_0, \boldsymbol{\epsilon}} \left[\frac{\beta_t^2}{2\sigma_t^2 \alpha_t (1 - \bar{\alpha}_t)} \|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \boldsymbol{\epsilon}, t)\|^2 \right]$$

Ho et al. (2020) osservarono che ignorare il peso $\beta_t^2/(\dots)$ e prendere una media uniforme su tutti i t (semplificazione $\mathcal{L}_{\text{simple}}$) funziona meglio in pratica.

APPENDICE A

Glossario Completo dei Simboli

Ogni simbolo matematico usato nel documento, organizzato per categoria.

A.1 Insiemi e Spazi

Simbolo	Significato	Esempio
$\mathbb{R}$	Numeri reali	$\pi \in \mathbb{R}$
$\mathbb{R}^n$	Spazio vettoriale reale di dimensione n	$\mathbf{x} \in \mathbb{R}^{512}$
$\mathbb{R}^{m \times n}$	Spazio delle matrici reali $m \times n$	$\mathbf{W} \in \mathbb{R}^{128 \times 64}$

$\mathbb{N}$	Numeri naturali $\{0, 1, 2, \dots\}$	$L \in \mathbb{N}$
$[N]$	Insieme $\{1, 2, \dots, N\}$	$i \in [N]$
$\mathcal{D}$	Dataset di training	$\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^N$
$\mathcal{H}$	Classe di ipotesi (insieme di modelli)	$h \in \mathcal{H}$
$\mathcal{N}(\mu, \sigma^2)$	Distribuzione gaussiana con media μ e varianza σ^2	$w \sim \mathcal{N}(0, 1)$
$\mathcal{U}[a, b]$	Distribuzione uniforme su $[a, b]$	$\theta \sim \mathcal{U}[-0.1, 0.1]$

A.2 Vettori e Matrici

Simbolo	Significato	Dimensione
$\mathbf{x}, \mathbf{y}, \mathbf{z}$	Vettori colonna (minuscolo grassetto)	$n \times 1$
$\mathbf{W}, \mathbf{A}, \mathbf{B}$	Matrici (maiuscolo grassetto)	$m \times n$
x_i	i -esima componente del vettore $\mathbf{x}$	scalare
W_{ij}	Elemento riga i , colonna j di $\mathbf{W}$	scalare
$\mathbf{x}^T$	Trasposta di $\mathbf{x}$ (vettore riga)	$1 \times n$
$\ \mathbf{x}\ _0$ $\ \mathbf{x}\ _2$	Norma euclidea: $\sqrt{\sum_i x_i^2}$	scalare ≥ 0
$\ \mathbf{x}\ _1$	Norma L1: $\sum_i x_i $	scalare ≥ 0
$\mathbf{x} \odot \mathbf{y}$	Prodotto di Hadamard (elemento per elemento)	$n \times 1$
$\mathbf{x}\mathbf{y}^T$	Prodotto esterno	$n \times m$
$\mathbf{I}_n$	Matrice identità $n \times n$	$n \times n$
$\text{diag}(\mathbf{v})$	Matrice diagonale con $\mathbf{v}$ sulla diagonale	$n \times n$

A.3 Calcolo e Ottimizzazione

Simbolo	Significato
$\frac{df}{dx}$ o $f'(x)$	Derivata di f rispetto a x
$\frac{\partial f}{\partial x_i}$	Derivata parziale di f rispetto a x_i (tutte le altre variabili fisse)
$\nabla_{\mathbf{x}} f$	Gradiente di f rispetto al vettore $\mathbf{x}$: vettore di tutte le derivate parziali
$\mathbf{J}_f$	Matrice Jacobiana di $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$: $(J_f)_{ij} = \partial f_i / \partial x_j$
$\nabla^2 f$ o $\mathbf{H}_f$	Matrice Hessiana: $H_{ij} = \partial^2 f / \partial x_i \partial x_j$
$\arg \min_x f(x)$	Il valore di x che minimizza f
$\theta \leftarrow \theta - \alpha \nabla \mathcal{L}$	Aggiornamento del gradiente (il simbolo $\leftarrow$ indica "assegnare")
α	Learning rate (tasso di apprendimento)
λ	Coefficiente di regolarizzazione
ε	Piccola costante numerica per stabilità (tipicamente 10^{-8})

A.4 Reti Neurali

Simbolo	Significato
L	Numero di layer della rete
$n^{[\ell]}$	Numero di neuroni nel layer ℓ

$\mathbf{W}^{[\ell]}, \mathbf{b}^{[\ell]}$	Pesi e bias del layer ℓ
$\mathbf{z}^{[\ell]}$	Pre-attivazione del layer ℓ : $\mathbf{W}^{[\ell]} \mathbf{a}^{[\ell-1]} + \mathbf{b}^{[\ell]}$
$\mathbf{a}^{[\ell]}$	Attivazione del layer ℓ : $f^{[\ell]}(\mathbf{z}^{[\ell]})$
$\delta^{[\ell]}$	Gradiente della loss rispetto alla pre-attivazione: $\partial \mathcal{L} / \partial \mathbf{z}^{[\ell]}$
$\hat{\mathbf{y}}$	Output predetto dalla rete
$\mathcal{L}(\theta)$	Funzione di loss parametrizzata da θ
θ	Insieme di tutti i parametri della rete
$\sigma(z)$	Funzione sigmoid: $1/(1 + e^{-z})$
$\text{softmax}(\mathbf{z})_k$	$e^{z_k} / \sum_j e^{z_j}$
δ_{ij}	Delta di Kronecker: 1 se $i = j$, 0 altrimenti
$\mathbf{1}[\cdot]$	Funzione indicatrice: 1 se la condizione è vera, 0 altrimenti

A.5 Probabilità e Statistica

Simbolo	Significato
$P(A)$	Probabilità dell'evento A
$P(A B)$	Probabilità condizionata di A dato B
$p(\mathbf{x})$	Densità di probabilità della variabile continua $\mathbf{x}$
$\mathbb{E}[X]$	Valore atteso (media) della variabile aleatoria X
$\mathbb{E}_p[f(X)]$	Valore atteso di $f(X)$ sotto la distribuzione p
$\text{Var}(X)$	Varianza: $\mathbb{E}[(X - \mathbb{E}[X])^2]$
$H(X)$	Entropia di Shannon di X
$H(p, q)$	Cross-entropy tra distribuzioni p e q

$$\cos(\alpha \pm \beta) = \cos \alpha \cos \beta \mp \sin \alpha \sin \beta$$

$$\sin^2 \theta + \cos^2 \theta = 1, \quad \cos(2\theta) = \cos^2 \theta - \sin^2 \theta = 1 - 2 \sin^2 \theta$$

$$e^{i\theta} = \cos \theta + i \sin \theta \quad (\text{formula di Eulero})$$

$$\text{Rotazione di } \theta : \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

B.3 Derivate Fondamentali

Funzione $f(x)$	Derivata $f'(x)$	Note
c (costante)	0	
x^n	nx^{n-1}	Regola della potenza
e^x	e^x	Unica funzione uguale alla sua derivata
$e^{f(x)}$	$e^{f(x)} \cdot f'(x)$	Regola della catena
$\ln x$	$1/x$ per $x > 0$	
$\ln f(x)$	$f'(x)/f(x)$	
$\sigma(x)$	$\sigma(x)(1 - \sigma(x))$	Sigmoid
$\tanh(x)$	$1 - \tanh^2(x)$	
$\max(0, x)$	$\mathbf{1}[x > 0]$	ReLU
$ x $	$\text{sign}(x)$ per $x \neq 0$	
$f(x)g(x)$	$f'(x)g(x) + f(x)g'(x)$	Regola del prodotto
$f(x)/g(x)$	$(f'g - fg')/g^2$	Regola del quoziente
$f(g(x))$	$f'(g(x)) \cdot g'(x)$	Regola della catena

B.4 Algebra Lineare

$$\begin{aligned}(\mathbf{AB})^T &= \mathbf{B}^T \mathbf{A}^T, & (\mathbf{AB})^{-1} &= \mathbf{B}^{-1} \mathbf{A}^{-1} \\ \text{tr}(\mathbf{AB}) &= \text{tr}(\mathbf{BA}), & \text{tr}(\mathbf{A}) &= \sum_i \lambda_i \text{ (somma autovalori)} \\ \det(\mathbf{AB}) &= \det(\mathbf{A}) \det(\mathbf{B}), & \det(\mathbf{A}^T) &= \det(\mathbf{A}) \\ \frac{\partial}{\partial \mathbf{x}} (\mathbf{a}^T \mathbf{x}) &= \mathbf{a}, & \frac{\partial}{\partial \mathbf{x}} (\mathbf{x}^T \mathbf{A} \mathbf{x}) &= (\mathbf{A} + \mathbf{A}^T) \mathbf{x} \\ \frac{\partial}{\partial \mathbf{X}} \text{tr}(\mathbf{AX}) &= \mathbf{A}^T, & \frac{\partial}{\partial \mathbf{X}} \ln \det(\mathbf{X}) &= (\mathbf{X}^{-1})^T\end{aligned}$$

B.5 Probabilità

$$\begin{aligned}P(A \cup B) &= P(A) + P(B) - P(A \cap B) \\ P(A|B) &= \frac{P(B|A)P(A)}{P(B)} \quad (\text{Teorema di Bayes}) \\ \text{Var}(X + Y) &= \text{Var}(X) + \text{Var}(Y) + 2\text{Cov}(X, Y) \\ \text{Var}(aX) &= a^2 \text{Var}(X), & \mathbb{E}[aX + b] &= a\mathbb{E}[X] + b \\ \text{se } X, Y \text{ indipendenti: } & \mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y] \\ \text{Legge dei grandi numeri: } & \frac{1}{N} \sum_{i=1}^N X_i \xrightarrow{N \rightarrow \infty} \mathbb{E}[X] \\ \text{Teorema del limite centrale: } & \frac{\sum_i X_i - N\mu}{\sigma\sqrt{N}} \xrightarrow{d} \mathcal{N}(0, 1)\end{aligned}$$

B.6 Teoria dell'Informazione

$$H(X) = - \sum_x p(x) \log p(x) \geq 0$$

$$H(X) \leq \log |\mathcal{X}| \quad (\text{massimo con distribuzione uniforme})$$

$$H(X, Y) = H(X) + H(Y|X) = H(Y) + H(X|Y)$$

$$I(X; Y) = H(X) - H(X|Y) = H(Y) - H(Y|X) \geq 0$$

$$D_{\text{KL}}(p||q) \geq 0, \quad D_{\text{KL}}(p||q) = 0 \iff p = q$$

$$H(p, q) = H(p) + D_{\text{KL}}(p||q) \geq H(p)$$

APPENDICE C

Complessità Computazionale delle Operazioni Principali

C.1 Operazioni di Base

Operazione	Tempo	Memoria	Note
Prodotto scalare $\mathbf{u}^T \mathbf{v}, \mathbf{u}, \mathbf{v} \in \mathbb{R}^n$	$O(n)$	$O(1)$	n moltiplicazioni, $n - 1$ somme
Prodotto $\mathbf{W}\mathbf{x}$, $\mathbf{W} \in \mathbb{R}^{m \times n}$	$O(mn)$	$O(m)$	m prodotti scalari
Prodotto $\mathbf{A}\mathbf{B}$, $\mathbf{A} \in \mathbb{R}^{m \times k}$, $\mathbf{B} \in \mathbb{R}^{k \times n}$	$O(mkn)$	$O(mn)$	In pratica sfrutta BLAS/cuBLAS

Inversione $\mathbf{A}^{-1}$, $\mathbf{A} \in \mathbb{R}^{n \times n}$	$O(n^3)$	$O(n^2)$	LU decomposition
SVD di $\mathbf{A} \in \mathbb{R}^{m \times n}$	$O(mn^2)$ per $m \geq n$	$O(mn)$	
Softmax su $\mathbf{z} \in \mathbb{R}^K$	$O(K)$	$O(K)$	Un passaggio per max, uno per exp, uno per normalizzare
Layer Norm su $\mathbf{x} \in \mathbb{R}^d$	$O(d)$	$O(d)$	Due passaggi: media e varianza

C.2 Layer della Rete

Layer	Parametri	FLOPs (forward)	Memoria attivazioni
FC: $n_{\text{in}} \rightarrow n_{\text{out}}$	$n_{\text{in}}n_{\text{out}} + n_{\text{out}}$	$2n_{\text{in}}n_{\text{out}}$	n_{out}
Conv2D: $C_{\text{in}}, C_{\text{out}},$ kernel k	$C_{\text{in}}C_{\text{out}}k^2$	$2C_{\text{in}}C_{\text{out}}k^2H_{\text{out}}W_{\text{out}}$	$C_{\text{out}}H_{\text{out}}W_{\text{out}}$
Self- Attention: seq n , dim d	$4d^2$	$8nd^2 + 4n^2d$	n^2 (matrice attenzione)
FFN: dim d , $d_{\text{ff}} = 4d$	$8d^2$	$16nd^2$	nd_{ff}
Transformer layer completo	$12d^2$	$24nd^2 + 4n^2d$	$O(nd + n^2)$

Embedding	$ V \cdot d$	$O(1)$ (lookup)	d
	$ V \rightarrow d$		

I FLOPs del backward pass sono tipicamente $2\times$ quelli del forward (una per il gradiente rispetto all'input, una per il gradiente rispetto ai pesi).

C.3 Scaling Laws — Come Cresce il Costo

Per un Transformer con L layer, dimensione d , vocabolario $|V|$ e sequenze di lunghezza n :

$$|\theta| \approx 12Ld^2 + |V| \cdot d \quad (\text{parametri totali})$$

$$\text{FLOPs per token (forward)} \approx 2|\theta| \quad (\text{empiricamente})$$

$$\text{FLOPs totali training} \approx 6|\theta| \cdot N_{\text{tokens}} \quad (\text{forward} + \text{backward} + \text{embedding})$$

Le **Chinchilla Scaling Laws** (Hoffmann et al., 2022) indicano il rapporto ottimale: per C FLOPs di budget, il modello ottimale ha circa $N^* \approx \sqrt{C/6}$ parametri addestrati su $D^* \approx \sqrt{6C}$ token.

APPENDICE D

Iperparametri Tipici e Regole Pratiche

D.1 Ottimizzatori

Iperparametro	Valore tipico	Note
Learning rate α (AdamW)	$1-3 \times 10^{-4}$	Dipende fortemente dall'architettura
β_1 (Adam/AdamW)	0.9	Quasi sempre lasciato al default
β_2 (Adam/AdamW)	0.95–0.999	0.95 per LLM, 0.999 per CV
ε (Adam)	10^{-8} – 10^{-6}	Aumentare per stabilità su float16
Weight decay λ (AdamW)	0.01–0.1	0.1 per LLM, 0.01 per CV
Gradient clip norm	1.0	Applica a quasi tutti i modelli profondi
Warmup steps	1–5% del training	Critico per Adam; evita instabilità iniziale

D.2 Architettura

Componente	Valore/Scelta tipica	Motivazione
Attivazione hidden	GELU o SwiGLU	Meglio di ReLU in quasi tutti i benchmark
Normalizzazione	RMSNorm (LLM), LayerNorm (Transformer), BN (CNN)	
Posizionamento LN	Pre-LN (prima del sublayer)	Training più stabile di Post-LN
Dropout rate	0.0–0.1 (LLM), 0.1–0.5 (MLP/CNN)	I LLM grandi raramente usano dropout

Batch size	512–32,768 token (LLM), 32–512 immagini (CNN)	Batch grandi richiedono LR più alto
d_{ff}/d (FFN)	$4\times$ (standard), $\frac{8}{3}\times$ (SwiGLU)	SwiGLU usa 3 matrici anziché 2
d_k (attention)	$d/h = 64\text{--}128$	Tipicamente 64 per la maggior parte dei modelli

D.3 Segnali Diagnostici Durante il Training

Segnale	Causa probabile	Rimedio
Loss NaN o Inf	LR troppo alto, overflow float16	Riduci LR; aggiungi gradient clipping; usa ε più grande in Adam
Loss scende lentamente all'inizio	LR troppo basso o warmup troppo lungo	Aumenta LR; riduci warmup
Loss oscilla molto	Batch size piccola o LR troppo alto	Aumenta batch; riduci LR; usa gradient accumulation
Training loss bassa, val loss alta	Overfitting	Aumenta dropout/weight decay; più dati; riduci modello
Gradienti molto piccoli	Vanishing gradient; LR troppo basso	Controlla inizializzazione; usa residual connections
Gradienti molto grandi	Exploding gradient	Gradient clipping; riduci LR

Riferimenti Bibliografici

Opere fondamentali e articoli originali che hanno introdotto i concetti trattati nel documento.

Libri di Testo

TESTI FONDAMENTALI

Goodfellow, I., Bengio, Y., Courville, A. (2016). *Deep Learning*. MIT Press. Disponibile gratuitamente su deeplearningbook.org. Il testo di riferimento per i fondamenti teorici delle reti neurali profonde.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer. Trattazione probabilistica e bayesiana dell'apprendimento automatico.

Murphy, K. P. (2022). *Probabilistic Machine Learning: An Introduction*. MIT Press. Prospettiva moderna con enfasi su modelli probabilistici.

Sutton, R. S., Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (2a ed.). MIT Press. Il testo canonico sull'apprendimento per rinforzo.

Hamilton, W. L. (2020). *Graph Representation Learning*. Morgan & Claypool. Introduzione completa ai metodi di apprendimento su grafi.

Articoli Fondativi

ARCHITETTURE E ALGORITMI BASE

Rumelhart, D. E., Hinton, G. E., Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323, 533–536. Prima dimostrazione pratica della backpropagation.

LeCun, Y., Bottou, L., Bengio, Y., Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278–2324. Introduzione delle CNN e applicazione all'MNIST.

Hochreiter, S., Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8), 1735–1780. Articolo originale dell'LSTM.

Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y. (2014). Learning phrase representations using RNN encoder–decoder for statistical machine translation. *arXiv:1406.1078*. Introduzione del GRU.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., Polosukhin, I. (2017). Attention is all you need. *NeurIPS 2017*. Architettura Transformer originale.

He, K., Zhang, X., Ren, S., Sun, J. (2016). Deep residual learning for image recognition. *CVPR 2016*. ResNet e residual connections.

Ottimizzazione

OTTIMIZZATORI E TEORIA

Kingma, D. P., Ba, J. (2015). Adam: A method for stochastic optimization. *ICLR 2015*. Introduzione dell'ottimizzatore Adam.

Duchi, J., Hazan, E., Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *JMLR*, 12, 2121–2159. AdaGrad.

Loshchilov, I., Hutter, F. (2019). Decoupled weight decay regularization. *ICLR 2019*. AdamW — Adam con weight decay disaccoppiato.

Glorot, X., Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. *AISTATS 2010*. Xavier initialization.

He, K., Zhang, X., Ren, S., Sun, J. (2015). Delving deep into rectifiers. *ICCV 2015*. He initialization e PReLU.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv:1707.06347*. PPO.

Normalizzazione e Regolarizzazione

TECNICHE DI REGOLARIZZAZIONE

Ioffe, S., Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *ICML 2015*. Batch Normalization.

Ba, J. L., Kiros, J. R., Hinton, G. E. (2016). Layer normalization. *arXiv:1607.06450*. Layer Normalization.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *JMLR*, 15(1), 1929–1958. Dropout.

Zhang, B., Sennrich, R. (2019). Root mean square layer normalization. *NeurIPS 2019*. RMSNorm.

Wu, Y., He, K. (2018). Group normalization. *ECCV 2018*. Group Normalization.

Modelli Generativi

VAE, GAN, DIFFUSION

Kingma, D. P., Welling, M. (2014). Auto-encoding variational bayes. *ICLR 2014*. VAE con reparametrization trick.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y. (2014). Generative adversarial nets. *NeurIPS 2014*. GAN originale.

Ho, J., Jain, A., Abbeel, P. (2020). Denoising diffusion probabilistic models. *NeurIPS 2020*. DDPM — diffusion models.

Song, Y., Sohl-Dickstein, J., Kingma, D. P., Kumar, A., Ermon, S., Poole, B. (2021). Score-based generative modeling through stochastic differential equations. *ICLR 2021*. Unificazione dei modelli di score e diffusion.

Architetture Moderne e Scaling

LLM E ARCHITETTURE RECENTI

Brown, T. B. et al. (2020). Language models are few-shot learners. *NeurIPS 2020*. GPT-3 e few-shot learning.

Touvron, H. et al. (2023). LLaMA: Open and efficient foundation language models. *arXiv:2302.13971*. LLaMA — modello efficiente con RMSNorm, SwiGLU, RoPE.

Su, J., Lu, Y., Pan, S., Murtadha, A., Wen, B., Liu, Y. (2021). RoFormer: Enhanced transformer with rotary position embedding. *arXiv:2104.09864*. RoPE.

Dao, T., Fu, D. Y., Ermon, S., Rudra, A., Ré, C. (2022). FlashAttention: Fast and memory-efficient exact attention with IO-awareness. *NeurIPS 2022*. Flash Attention.

Hoffmann, J. et al. (2022). Training compute-optimal large language models. *arXiv:2203.15556*. Chinchilla scaling laws.

Fedus, W., Zoph, B., Shazeer, N. (2022). Switch transformers: Scaling to trillion parameter models. *JMLR*, 23(1). Mixture of Experts per i Transformer.

Ainslie, J. et al. (2023). GQA: Training generalized multi-query transformer models from multi-head checkpoints. *EMNLP 2023*. Grouped Query Attention.

Teoria e Generalizzazione

FONDAMENTI TEORICI

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*,

2(4), 303–314. Teorema di approssimazione universale.

Vapnik, V., Chervonenkis, A. (1971). On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and Its Applications*, 16(2). Teoria VC originale.

Belkin, M., Hsu, D., Ma, S., Mandal, S. (2019). Reconciling modern machine-learning practice and the classical bias–variance trade-off. *PNAS*, 116(32). Double descent.

Jacot, A., Gabriel, F., Hongler, C. (2018). Neural tangent kernel: Convergence and generalization in neural networks. *NeurIPS 2018*. NTK.

Neal, R. M. (1994). *Bayesian Learning for Neural Networks*. PhD thesis, University of Toronto. Reti neurali bayesiane e connessione GP.

Zhang, C., Bengio, S., Hardt, M., Recht, B., Vinyals, O. (2017). Understanding deep learning requires rethinking generalization. *ICLR 2017*. Le reti neurali memorizzano etichette casuali — riflessione sulla generalizzazione.

Graph e Metric Learning

GNN E APPRENDIMENTO METRICO

Kipf, T. N., Welling, M. (2017). Semi-supervised classification with graph convolutional networks. *ICLR 2017*. GCN.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y. (2018). Graph attention networks. *ICLR 2018*. GAT.

Hadsell, R., Chopra, S., LeCun, Y. (2006). Dimensionality reduction by learning an invariant mapping. *CVPR 2006*. Contrastive loss.

Chen, T., Kornblith, S., Norouzi, M., Hinton, G. (2020). A simple framework for contrastive learning of visual representations. *ICML 2020*. SimCLR e NT-Xent loss.

Schroff, F., Kalenichenko, D., Philbin, J. (2015). FaceNet: A unified embedding for face recognition and clustering. *CVPR 2015*. Triplet loss.

Matematica delle Reti Neurali — Guida Completa ed Esplicita

50 parti + 4 appendici · ogni derivazione passo per passo · ogni simbolo definito ·
esempi numerici espliciti